Survey of Texture Mapping Techniques for Representing and Rendering Volumetric Mesostructure

Charalampos Koniaris
University of Bath

Darren Cosker
University of Bath

Xiaosong Yang
University of Bournemouth

Kenny Mitchell
Disney Research

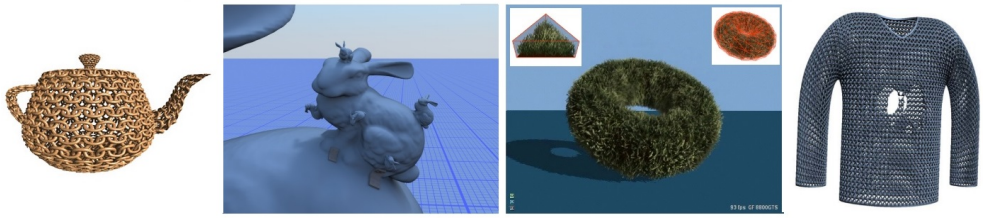


Figure 1. Volumetric texture mapping can be used to represent complex surface detail on parameterised surfaces. Figures from [Policarpo and Oliveira 2006], [Brodersen et al. 2007], [Decaudin and Neyret 2009] and [Peng et al. 2004].

Abstract

Representation and rendering of volumetric mesostructure using texture mapping can potentially allow the display of highly detailed, animated surfaces at a low performance cost. Given the need for consistently more detailed and dynamic worlds rendered in real-time, volumetric texture mapping now becomes an area of great importance.

In this survey, we review the developments of algorithms and techniques for representing volumetric mesostructure as texture-mapped detail. Our goal is to provide researchers with an overview of novel contributions to volumetric texture mapping as a starting point for further research and developers with a comparative review of techniques, giving insight into which methods would be fitting for particular tasks.

We start by defining the scope of our domain and provide background information regarding mesostructure and volumetric texture mapping. Existing techniques are assessed in terms of content representation and storage as well as quality and performance of parameterization and rendering. Finally, we provide insights to the field and opportunities for research directions in terms of real-time volumetric texture-mapped surfaces under deformation.

 ISSN 2331-7418

(a) Sponge (b) Cloth (c) Tree (d) Bread

Figure 2. Examples of volumetric mesostructures that are difficult to represent accurately with current interactive rendering techniques.

1. Introduction

Rich visual detail in computer-generated imagery has always been a great challenge in computer graphics. *Texture mapping*, pioneered by [Catmull 1974], has been one of the most successful such techniques with a vast number of applications, one of the most common being mapping *mesostructure* (fine scale detail) on surfaces using 2D parameterization.

1.1. Problem Domain and Applications

Volumetric (3D) mesostructure is a class of mesostructure that allows detail of arbitrary complexity on a surface (Figure 1) at the expense of rendering performance, storage costs, and ease of authoring compared to 2D mesostructure. Examples of volumetric mesostructure include holes (Figures 2(a), 2(d)), cloth (Figure 2(b)) or tree bark (Figure 2(c)). While alternative representations (voxels, point clouds) can capture such complexity by treating surface detail as part of the surface, animating or deforming such representations becomes very difficult, such as tree bark bending in the wind or a deforming realistic cloth. Even if performance and storage costs are not a concern, animating models with complex volumetric detail using alternative representations is a very difficult and time-consuming task. Volumetric texture mapping provides abstraction of detail from the animating surface, allowing the use of efficient animation methods for the underlying surface without inducing any extra costs due to animation in terms of authoring effort or storage. Additionally, volumetric representation of texture detail allows pre-filtering of the represented shape in multiple resolutions (e.g., [Crassin et al. 2009]), therefore solving problems of explicit geometric representations such as level-of-detail and aliasing.

Additional applications which can leverage the advantages of volumetric texture mapping include volumetric virtual makeup on characters and translucent surface detail on deformable models. (Figure 3).

1.2. Scope

In this survey, we review techniques that contribute to volumetric texture mapping. Contrary to other surveys in well-established and specialized areas of research in

Figure 3. Fantasy character with volumetric surface detail, courtesy of deviantART artist Pablo Andreetta.

mesostructure rendering (e.g., GPU displacement mapping [Szirmay-Kalos and Umenhoffer 2008], bidirectional texture functions (BTFs) [Filip and Haindl 2009], procedural noise functions [Lagae et al. 2010]), we set our scope to encompass techniques that share the goal of representing and rendering volumetric detail using texture mapping.

The gap in the current literature and the reasons for the defined scope become apparent if we look into the intersection of *real-time rendering*, *volumetric mesostructure* and *deformable surfaces*. Real-time animation of 3D models is typically done using *skeletal animation* on polygonal or subdivision surfaces. Such surfaces are relatively sparse and use texture and displacement mapping for increased visual complexity. Texture detail is typically mapped from a 2D texture to a surface. Similarly, volumetric detail is mapped from a 3D texture to a volume. As this extension is nontrivial within the context of real-time performance and the given scope, we split it into subprocesses and identify in which process each belongs in a volumetric texturing pipeline:

- Detail representation. (Content representation/storage)
- Definition of the volume as a function of the surface (Surface parameterization)
- Mapping of 3D texture detail to the volume (Mapping)
- Rendering of detail in the volume (Rendering)

We structure this survey by reviewing techniques that contribute to *content representation and storage*, *mapping*, and *rendering* in the context of volumetric mesostructure. As animation performance is a major reason for using volumetric texture map-

Technique	3. **Storage**			4. **Mapping**	5. **Rendering**		
	Procedural	Texture			Raycasting	Lookups	Slice-based
		Dense	Compressed				
[Wang et al. 2003] (**VuDm**)			✓			✓	
[Wang et al. 2004] (**GnDm**)			✓			✓	
[Peng et al. 2004] (**AvDf**)				✓			✓
[Hirche et al. 2004]				✓			
[Wang et al. 2005] (**MsDf**)			✓			✓	
[Donnelly 2005] (**DfDm**)					✓		
[Porumbescu et al. 2005] (**Sm**)				✓	✓		
[Dufort et al. 2005] (**STSm**)				✓	✓		
[Policarpo and Oliveira 2006] (**MLRm**)		✓			✓		
[Zhou et al. 2006] (**StreMe**)				✓			
[Ritsche 2006] (**RTSS**)					✓		
[Jeschke et al. 2007] (**SSm/CSm**)				✓	✓		
[Brodersen et al. 2007] (**LaTraP**)				✓			
[Decaudin and Neyret 2009] (**VolB**)				✓			✓
[Gilet and Dischler 2009]	✓						

Table 1. Overview of novel contributions of reviewed techniques to a volumetric texture rendering pipeline. The abbreviations are used in the rest of the paper, mainly Section 6.

ping, techniques are discussed in terms of their characteristics and potential for real-time rendering and application to deformable surfaces (Table 1).

While many of the techniques support advanced lighting and shading effects in varying degrees, we choose to only focus on mesostructure *geometry* rather than *appearance*. So, this excludes discussion of effects such as shadows, reflections, global illumination, and sub-surface scattering, although transparency is included as it has an effect on the perceived shape. Additionally, this survey is not about general rendering or sampling of volume data, as we choose to focus on contributions specific to volume data rendering using texture mapping.

1.3. Comparison Measures

Our evaluation of the reviewed techniques is based on implementations (ours or acquired), as well as the original papers. Our implementations were developed for visual comparison purposes, and, as such, they may not contain all optimizations mentioned in the papers. We developed the “building blocks" of several techniques, such as shell rendering using the triangulated shell silhouette (GnDm, SSm, CSm), shell rendering using tetrahedra (Sm, STSm, RTSS), view-space slice-based rendering (VolB), raycasting curved ray paths in texture space (Sm, SSm, CSm), raycasting acceleration using distance fields (DfDm, RTSS, SSm, CSm) and used an existing implementation for MLRm. Table 1 lists all the methods we compare in our survey, along with associated acronyms for readability purposes.

Comparisons of storage requirements do not require actual implementations, while coarse performance measurements can be derived by the complexity of techniques. For example, extruding a point towards a normal is a single shader operation and thus takes microseconds when applied on all points of an average model, while solving non-linear optimization problems on these points is typically an offline operation. As such, our unimplemented subjective evaluations are restricted to coarse comparisons only, to avoid misguiding the reader.

1.4. Overview

We first present essential background information that introduces the problem (Section 2). Techniques are then reviewed in terms of their contributions to:

- Content representation and storage (Section 3);
- Mapping of 3D mesostructure space on surfaces (Section 4);
- Rendering of volumetric data (Section 5).

We then discuss and compare rendering techniques in terms of rendering artifacts and limitations (Section 6). We finally summarize and provide concluding remarks (Section 7).

2. Background

In this section, we introduce essential concepts regarding volumetric mesostructure and its mapping on surfaces. Mesostructure is defined first as a range in terms of geometric scale, followed by a brief introduction to the coordinate spaces used for mapping the detail on surfaces. We then list common representations of mesostructure detail, based on the complexity of detail they represent. Finally, base surface representations that are typically used in texture mapping are briefly introduced.

2.1. Geometric Scale

Geometric detail can be split into three main categories in terms of scale: *macro*, *meso* and *micro* [Westin et al. 1992; Koenderink and van Doorn 1996]. Macroscopic geometry (or *macrostructure*) is considered geometry of large structures and is usually represented with standard geometric primitives, such as polygons or parametric patches. Mesoscopic detail (or *mesostructure*), is high-frequency detail on object surfaces, distinguishable from a close distance. Such detail is usually stored separately in *texture space* and is mapped on macroscopic surfaces. Microscopic detail (or *microstructure*) is invisible to the human eye, so instead of being explicitly represented, its light scattering properties are typically modeled using *bi-directional reflectance distribution functions* (BRDFs) [Glassner 1994].

2.2. Texture, Tangent, and Shell Space

Let us first define texture space as the coordinate system in which texture data are expressed. Coordinates usually span the range [0,1], and are typically 1D, 2D, or 3D. We denote these coordinates by (u,v,w). Let $\vec{S}(u,v)$ be a 2D-parameterised surface, $S:\mathbb{R}^2 \rightarrow \mathbb{R}^3$. The tangent space on the surface is defined by the tangent, bitangent, and normal vectors:

$$\vec{T}(u,v) = \frac{\partial S}{\partial u}$$
$$\vec{B}(u,v) = \frac{\partial S}{\partial v}$$
$$\hat{N}(u,v) = \frac{\vec{T}(u,v) \times \vec{B}(u,v)}{\left\| \vec{T}(u,v) \times \vec{B}(u,v) \right\|}$$

If we normalize $\vec{T}(u,v)$ and $\vec{B}(u,v)$, we can obtain a matrix that converts points from texture space to object space:

$$\mathbf{TBN}(u,v) = \begin{bmatrix} \vec{T}(u,v) \\ \vec{B}(u,v) \\ \vec{N}(u,v) \end{bmatrix}$$

More information about tangent space can be found in [Szirmay-Kalos and Umenhoffer 2008].

Shell space can be defined as a variably thick layer over the base surface:

$$\vec{G}(u,v,w) = \vec{S}(u,v) + wH(u,v)\hat{d}(u,v)$$

where $H(u,v)$ is defined as the shell scalar thickness per surface point and $\hat{d}(u,v)$ is a point on the unit sphere, typically the surface normal $\hat{N}(u,v)$. In practice, the function $H(u,v)$ is either per-vertex interpolated or reduced to a single constant value H to yield simpler computations. Shell space and mapping can be seen in Figure 4.

2.3. Mesostructure Complexity

In terms of appearance, surface mesostructure can be either opaque or translucent. Opaque detail is more efficient to store and render, but can only represent a subset of representable materials. In terms of shape and structure, surface mesostructure can be broken down into three categories, in ascending order of representable complexity: height fields, vector fields, and density fields.

- *Height fields* represent shape by "pushing" each point on a base surface towards the surface normal. Given a height field $h(u,v)$, we can compute the mesostruc-

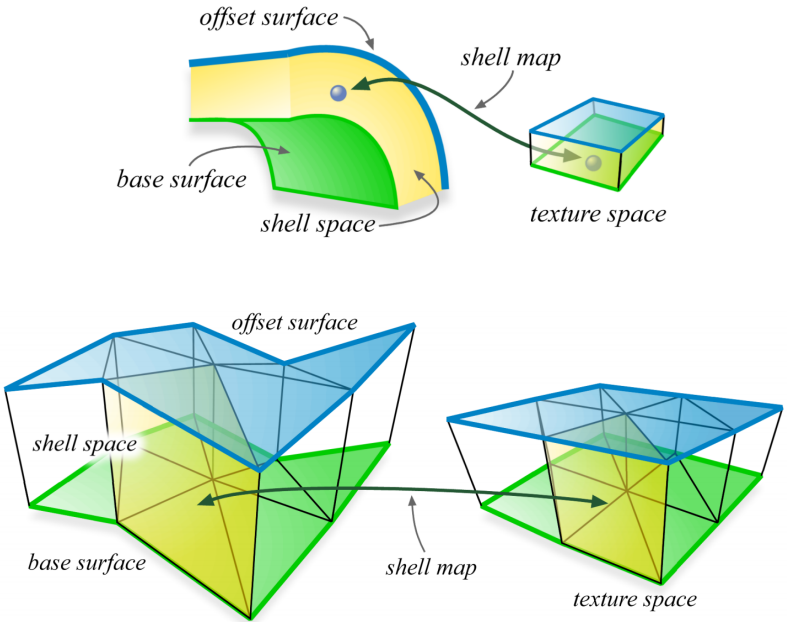


Figure 4. With shell mapping, 3D texture detail is mapped inside a thick layer over the surface of a model. Figures from [Porumbescu et al. 2005].

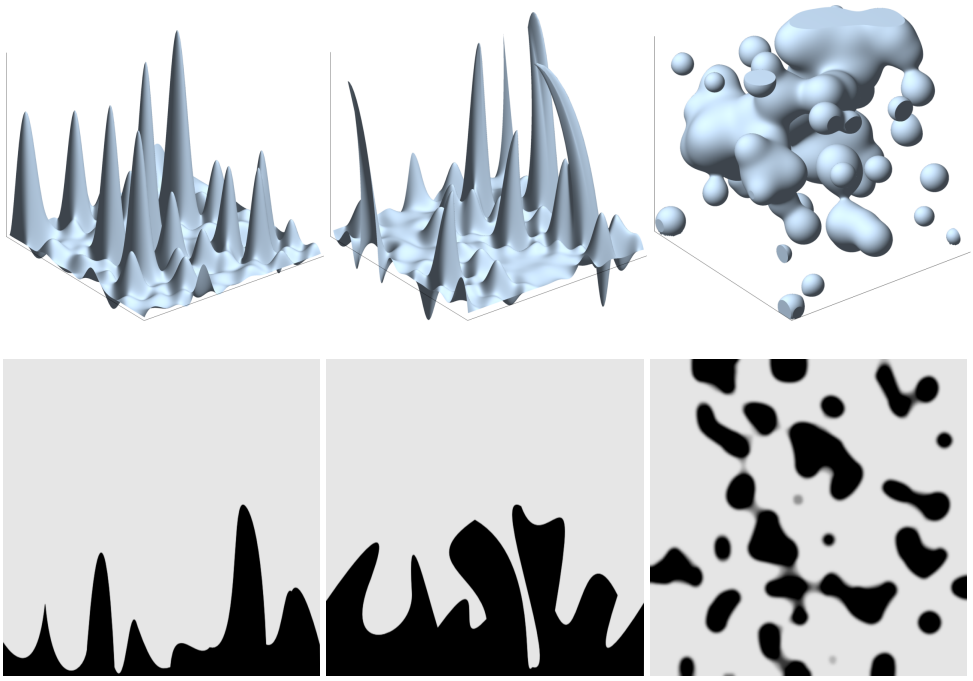
ture point in object space as

$$\vec{p}(u,v) = \vec{S}(u,v) + h(u,v)\hat{N}(u,v)$$

- *Vector fields* represent shape by "pushing" each point on a base surface towards an arbitrary direction. Shape and structure representations using height fields are a subset of the possible representations using vector fields. Given $\vec{d}(u,v)$, $d : \mathbb{R}^2 \to \mathbb{R}^3$ as the mesostructure vector displacements in texture space, we can compute the mesostructure point in object space as

$$\vec{p}(u,v) = \vec{S}(u,v) + \mathbf{TBN}(u,v)\vec{d}(u,v)$$

  Here, the **TBN** matrix is used to transform the vector displacements from texture space to object space.
- *Density fields* represent shape by explicitly assigning densities on points in a volume around a base surface. Shape and structure representations using vector fields are a subset of the possible representations using density fields. We can sample the density of any point on or above the macrostructure surface directly from a given density field $D(u,v,w)$ mapped on the surface.



(a) Height field (b) Vector field (c) Density field

Figure 5. Mesostructure representation complexity.

Figure 5 shows examples of these categories.

2.4. Surface Representations for Texture Mapping

Texture mapping is a common application of mesh parameterization algorithms [Hormann et al. 2007]. Commonly used surface representations for such parameterizations are polygonal meshes, NURBS, and subdivision surfaces. While polygonal meshes are currently the most widely used format for real-time rendering, subdivision surfaces have been a common modelling primitive used by artists, especially for animation and offline rendering [DeRose et al. 1998]. Support for fine detail on such primitives allows easily animated complex surfaces [Lee et al. 2000; Nießner and Loop 2013]. Recent graphics hardware and APIs have added support for dynamic tessellation among other features and have enabled rendering of subdivision surfaces in real-time [Loop et al. 2009; Kovacs et al. 2009; Nießner et al. 2012].

Ptex, introduced by [Burley and Lacewell 2008] is a texture-mapping method for quad-based subdivision surfaces that does not require explicit parameterization. While it was originally used for production rendering, McDonald and Burley recently improved its performance to real-time using commodity Direct3D 11 hardware [McDonald and Burley 2011].

Representation		Storage cost	Evaluation complexity	Sampling cost
Procedural 3.1		★★★★	★☆☆	★★☆
Textures	Dense 3.2.1	★☆☆☆	★★★	★★★
	Hierarchical 3.2.2	★★★☆	★★☆	★★★
	Compressed 3.2.3	★★☆☆	★☆☆	★★☆
Geometry 3.3		★★☆☆	★★★	★☆☆

Table 2. Relative comparison of representations for storing mesostructure data in terms of storage, evaluation, and filtering costs. For more details about the categories and the ratings, refer to Section 3.4

3. Content and Storage

Surface detail can be stored in three forms: as procedural definitions, as texture maps, or as actual geometry. Each representation has to balance storage, evaluation, and sampling costs (Table 2). The importance of each of these characteristics for real-time rendering varies depending on the application, e.g., high storage cost can create bottlenecks in hardware where video memory bandwidth is limited.

Below, we discuss representations based on the above categorization: procedural, textures, and geometry, and note their support for level-of-detail (LOD) and filtering. We summarize the section by providing a relative comparison of techniques.

3.1. Procedural Content

After the introduction of texture mapping and volumetric textures, it became clear that authoring of complex and highly detailed volumetric mesostructure is tedious to do manually and is very costly in terms of storage. Procedural techniques can be used for representing both macrostructure and mesostructure detail, and they have traditionally been used for both.

Procedural noise functions were originally introduced by [Perlin 1985] as a means to create controlled stochastic effects. [Ebert et al. 2002] expanded procedural functions to model more natural phenomena and structures. These functions became a very popular method of generating complex mesostructure detail because of their efficiency and negligible storage costs.

[Perlin and Hoffert 1989] used such functions in order to generate and display volumetric mesostructure (*hypertexture*). Hypertexture rendering requires a volumetric density-based representation of the entire object, and it works by distorting the 3D space of the object using one or multiple *density modulation functions* (DMFs), which are functions that modulate the density of a given point in $\mathbb{R}^3$. An *object density function* (ODF) is a function that describes the density d of a 3D shape in $\mathbb{R}^3$, $d \in [0,1]$. Hypertexture is created by successive application of DMFs to an object's ODF, using function primitives such as *gain*, *bias*, *noise*, and *turbulence*, as well as arithmetic functions. These primitives are combined using arithmetic expressions, and the re-

sulting DMFs can be also combined with Boolean set operators. Surface colors are computed in the end, after the shape and densities have been modeled, using various techniques such as procedural texturing. Hypertexture was originally developed as an offline technique, but it has been implemented in GPUs in order to achieve real-time performance [Miller and Jones 2005].

A recent approach by [Gilet and Dischler 2009] revisits hypertexture modeling and attempts to improve usability at the cost of generality. Instead of using just a density function, an additional set of three transfer functions are introduced: shape, color, and transparency. The density function adjusts points based on a scaled 3D vector field. The scaling factor is a formula that is based on the shape function indexed by noise. The vector field and model parameters are used to make the model more intuitive to users, as fully procedural approaches are always more difficult to use in order to generate desired results. The technique's implementation is GPU-compatible by storing the vector field, as well as density and transfer functions in textures which are accessed when rendering with shaders.

Procedural functions have also been used to augment low-resolution detail. [Satherley and Jones 2002] use distance fields to apply hypertexture to complex volumetric data. [Kniss et al. 2002] use procedural functions to perturb existing low resolution volume data in order to model more interesting high-frequency detail.

Volumetric mesostructure detail can also be generated semi-procedurally using *texture synthesis*, which is the process of generating large textures similar to given (smaller) exemplar textures. Regarding volumetric mesostructure, synthesis algorithms have been developed for solid textures [Pietroni et al. 2010], for geometry [Bhat et al. 2004; Lai et al. 2005; Zhou et al. 2006], as well as for BTFs [Filip and Haindl 2009].

In the case of deforming models, standard hypertexture-based noise results in the undesirable effect of points on the surface corresponding to different noise values at different frames. To avoid this problem, surface noise can be used instead [Lagae et al. 2010].

Procedural descriptions of volumetric mesostructure have the benefit of low storage requirements, but they are difficult to control in order to generate specific results. Evaluation of such procedural definitions can be performance-intensive and is governed by the evaluation of noise functions, although the latter can be approximated with precalculated textures for improved performance. As such, they are preferred in hardware where video-memory bandwidth is low relative to GPU computational power. Procedural descriptions are better suited for stochastic mesostructure representation, or augmenting existing mesostructure with stochastic detail. Also, filtering is difficult for procedurally generated content, and methods need to balance accuracy (analytic integration of noise function) against performance (global fading and noise frequency clamping) [Lagae et al. 2010; Bruneton and Neyret 2012].

[Heitz et al. 2013] present an efficient filtering scheme for the special case of color mapping noise functions by computing and sampling on-the-fly specialized filtering distributions.

3.2. Texture Maps

Texture maps are the most common form of mesostructure representation. We can describe such maps in a generalized sense as discrete scalar or vector-valued functions in $\mathbb{R}^n$ space and categorize them as *dense*, *hierarchical*, and *compressed*, depending on the sparsity and organization of data.

3.2.1. Dense textures

Volume textures are a direct extension of 2D texture maps; they store values for all voxels in a uniform 3D grid. While this form is simple, straighforward, and very efficient to evaluate, it is also very costly in terms of storage. Such textures have been used in a variety of techniques to render volumetric mesostructure in real-time [Peng et al. 2004; Donnelly 2005; Dufort et al. 2005; Ritsche 2006; Jeschke et al. 2007; Decaudin and Neyret 2009], but the storage cost restricts mesostructure to repeating patterns of small volume textures over the macrostructure surface. The type of data in such maps can vary, from colors to surface normals, density or distance fields, etc.

[Wang et al. 2005] introduce the *mesostructure distance function* (MsDf), a 4D function that represents the mesostructure distance from the reference plane given a 2D position on the plane and a viewing direction. MsDf data are quantized to 8 bits per pixel, so they become small enough to store uncompressed in a 3D texture, packing lighting direction in the third dimension. Their high-dimensional nature makes higher resolution data prohibitive in terms of storage cost, unless compressed. As the function is not smooth, it is difficult to employ standard high-dimensional data reduction techniques such as SVD.

[Policarpo and Oliveira 2006] propose an extension of 2D *relief textures* [Oliveira et al. 2000] that can capture non-heightfield details with significantly reduced storage requirements. Their technique is a generalization of relief mapping that supports multiple layers (MLRm). As in relief maps, the required data are normal maps and depth maps, indexable by texture coordinates. The extension requires these maps for multiple depth layers. For example, for a four-layer relief map, they need three textures: an RGBA texture storing a depth layer per channel, and two RGBA textures for the surface normals. Each of the latter two textures stores, for each layer, one of the x- and y-components of unit-length normal; unit-length normals z-component can be retrieved by $z = \sqrt{1 - x^2 - y^2}$.

The technique relies on surfaces that can be represented with a small number of layers and performs well under that assumption, as the memory consumption and

computational costs are not significantly higher than normal relief mapping. The layer-number restriction makes generalization difficult and increases the cost for complex surfaces that require a large number of layers. Translucent mesostructure is not supported. Finally, the technique is described in this paper using a single color map, resulting in all depth layers sharing the same color. In order to avoid this restriction, additional color maps should be introduced, and the algorithm should be changed so that it samples the appropriate color map depending on the depth layer that is used. By taking this into account, for each of the four layers, seven textures are needed: a depth map, two normal maps, and four color maps.

Filtering for textures depends on the nature of the texture data. The most important development for both LOD and filtering was the invention of *mipmapping* by [Williams 1983], in which a texture is stored in a pyramid along with a number of pre-filtered versions of it (downscaled by a factor of two at each level), while at runtime the rendering algorithm switches between mip-levels or applies trilinear/quadrilinear (2D/3D textures) interpolation based on the distance of the viewer to the texture. Mipmapping works as a very fast sampling method for texture data, and it is also efficient in the use of texture-cache hardware.

While for linear data (e.g., color, heightfield, and density) averaging (for construction) and bilinear/trilinear filtering (for sampling) are usually good enough; other types of data cannot use such linear filtering methods. Surface normal maps cannot be averaged, since averaging of normal vectors changes the apparent roughness of the surface. A major group of methods that have been developed to overcome this issue are approximations of the *normal distribution function* (NDF) [Fournier 1992; Schilling 1997; Olano and North 1997; Tan et al. 2005; Toksvig 2005; Han et al. 2007; Olano and Baker 2010]. Yet a different method is to switch between algorithms depending on the required level of detail [Kajiya 1985; Becker and Max 1993]. [Bruneton and Neyret 2012] have recently surveyed pre-filtering methods for more non-linear data such as normal, horizon, shadow, and procedural maps.

3.2.2. Hierarchical textures

Volumetric texture data often contain a significant amount of uniform space (with empty or constant material properties). To avoid this redundant storage, hierarchical data structures such as octrees were employed in order to store detail only where needed, effectively exploiting them as a form of compression [Neyret 1998].

The concept was later expanded to 3D textured models and was used for 2D texturing of unparameterised 3D models [Benson and Davis 2002; DeBry et al. 2002]. These techniques use the object space position (rest pose for animated models) to sample from the octree texture. To avoid incorrect averaging of texture color from opposite-facing points at thin features, normal information is used for sampling.

Octrees remained a widely-used data structure with the advent of GPU programming, as they can be efficiently implemented on GPUs for 3D data storage and access [Lefebvre et al. 2005; Lefohn et al. 2006] for use with real-time rendering. In recent years, octrees have been used as a hierarchical representation of volumetric data for out-of-core rendering [Gobbetti et al. 2008; Crassin et al. 2009; Iglesias Guitián et al. 2010; Laine and Karras 2010], although, in all such cases, no distinction is made between macrostructure and mesostructure.

Octrees are natural candidates for a LOD/filtering scheme via their intermediate nodes, which act as lower mipmap levels, so they share filtering schemes with dense textures.

3.2.3. Compressed, high-dimensional textures

High-dimensional textures can represent volumetric materials by evaluating appearance or structure as a function of many variables, such as texture coordinates and view direction. The first such form, *bidirectional texture functions* (BTF), was introduced by [Dana et al. 1999] to accurately represent real-world surfaces. BTFs are 7D functions that model material appearance based on a 2D position on a plane (u, v), illumination (θ_i, ϕ_i) and viewing angles (θ_v, ϕ_v), and spectral index (r):

$$\mathrm{BTF}\,(u, v, r, \theta_i, \phi_i, \theta_v, \phi_v)$$

Samples of this function are initially acquired as 2D texture maps from real materials using specialized hardware. As the volume of data can be overwhelming, they are typically compressed either by calculating analytical BRDFs per-texel, or by reducing the dimensionality using linear factorization [Filip and Haindl 2009]. During rendering, at each point on the object's surface, the viewing and lighting directions are transformed to tangent space and used for sampling the function.

BTFs can capture volumetric mesostructure with a wide variety of complex lighting and shading effects, such as mesostructure interreflections, shadows, translucency, and subsurface scattering, but they are not suited for texturing animated models in real-time, because of their acquisition difficulty, storage cost, deformation artifacts, and lack of silhouettes and depth information.

As the goal of BTFs is to model material appearance, they are optimized for small patches of stochastic or periodic content that can be efficiently mapped on a bigger surface by seamless mapping or texture synthesis. This is different than the more general case of surface mesostructure that is non-homogeneous on a larger scale; besides the acquisition difficulty in the case of real-world materials, larger patches result in slower compression times, bigger storage requirements, and slower evaluation time.

Deformation can modify the local curvature or scale of the surface. Both of these changes lead to artifacts when sampling BTFs, as the spatial relationship of the volumetric mesostructure elements changes in object space, thus invalidating precalcu-

lated lighting/shading effects.

So, while BTFs in their original form are inadequate for more general volumetric mesostructure rendering, other techniques were developed that used the concept of high-dimensional texture functions to achieve real-time rendering of complex surface detail.

[Wang et al. 2003] use *view-dependent displacement mapping* (VuDm) to efficiently render surface mesostructure with silhouettes, using concepts from BTFs. VuDm stores displacements along the viewing direction using a 5D function. The function is parameterised on 2D texture coordinates on the reference plane, view direction as polar angles, and local curvature of the base surface. They also use a 4D *maximal view polar angle map* (MVM) which is parameterised in terms of the texture coordinates on the reference plane, the local curvature, and the azimuth angle of the view direction, and store the maximum polar angle for the given parameters. For compression, VuDm and MVM are packed into a 2D matrix on which they apply SVD, resulting in a set of 2D weight maps, a set of 2D eigen maps for VuDm, and a set of 2D eigen maps for MVM. The MVM is used to improve the SVD-based compression, as the mesostructure distances display high frequency near silhouettes, resulting in the need for more eigen maps for a good reconstruction. While, in theory, this technique can represent non-heightfield surfaces, the authors use it to represent heightmaps, as it results in calculating the actual 2D texture coordinates, which can be used to sample any other textures.

[Wang et al. 2004] suggest *generalized displacement maps* (GnDm) to improve on two important drawbacks of VuDm: surface curvature coupling to mesostructure and restriction to heightmap representation. A GnDm is a 5D function that represents the distance to solid mesostructure from any point within a volumetric texture and is parameterised on 3D shell texture coordinates and view direction. For compression, they use SVD to decompose into 2D eigen maps and 3D weight maps.

All the techniques described reduce the storage requirements posed by the original BTF formulation by sacrificing lighting and shading precomputations. In terms of storage, they still rely on mesostructure data that can be easily compressed using SVD-based methods. This lack of generality in terms of compression, along with the other drawbacks intrinsic to high-dimensional texture functions, makes these techniques impractical for use in texturing of animated models with complex and varied mesostructure.

LOD and filtering are more involved in BTF-based techniques, as not all parameters can be interpolated in the same way (e.g., linearly). In VuDm and GnDm, the authors suggest mipmapping the whole function for easy antialiasing, although that increases the storage cost even more. In MsDf, the 4D data are packed in a volume texture by flattening the view directions, so filtering is performed using trilinear interpolation.

3.3. Geometry

An alternative volumetric mesostructure representation (to textures and procedural definitions) is *geometric textures*, geometric primitives stored as an actual mesh in texture space and then mapped on the macrostructure surface [Elber 2005]. Similar to dense textures, such approaches suffer from poor scalability in terms of storage: for high-complexity mesostructure over a surface, the number of primitives can quickly become prohibitive for storage and real-time rendering, especially if LOD-ready representations are used, such as multi-resolution meshes [Eck et al. 1995].

While straightforward to render, geometric textures inherit the drawbacks of highly-detailed geometry: difficulty of generating LODs [Hu et al. 2010], inability to represent fully volumetric data, and inefficient rasterization of micro-geometry in modern GPU rasterization pipelines [Fatahalian et al. 2009; Fatahalian et al. 2010]. As such, these approaches are not generally used for real-time rendering.

3.4. Summary

Table 2 summarizes our observations from the reviewed volumetric texture representations. "Geometry" refers to regular meshes, since multi-resolution meshes have not been used for geometric texture detail and, as such, a comparison with other techniques would be hypothetical. The ratings are based on a consensus among the authors, and while they do not reflect implementation results, they serve as useful, high-level comparison measures.

Storage. We determine storage cost in terms of the storage required in GPU memory for visible data. Procedural functions offer the best compression, as they only store function parameters. Dense textures are the most wasteful in terms of storage, as they do not use any form of compression and store both useful and non-useful data (e.g., fully transparent). Hierarchical textures store only useful data and can be efficiently used with streaming, in to order to keep in GPU memory, only visible data at the required LOD. Compressed high-dimensional textures use significantly less memory compared to their equivalent dense variant, but the compression effectiveness depends on the complexity and smoothness of data. Geometric representations only store useful data, but have no efficient LOD schemes, so all data need to be in memory.

Evaluation. Evaluation complexity is defined as the computational cost to access, decompress. or evaluate the mesostructure data depending on their storage format. Dense textures and geometry are the simplest to evaluate. Hierarchical texture traversal is a quite efficient form of evaluation, while evaluation of SVD-compressed textures requires large multiplications and evaluation of procedural functions can involve complex calculations.

Sampling. Sampling cost is defined as the cost (or difficulty) to filter and sample the evaluated mesostructure data. Dense and hierarchical textures are very efficient in terms of filtering, using mipmaps and inner nodes, respectively. Compressed textures can use linear interpolation, but the non-linear nature of some of the parameters can result in artifacts. Filtering geometry and procedural functions for general cases are difficult and open problems.

4. Mapping

Volumetric detail defined in texture space can be either mapped on the volume defined by a thick shell over the base surface, or represented by mapping the "ceiling" of the volumetric detail space on the base surface, with the mesostructure variation appearing over and under the surface, respectively.

In this section, we first give some definitions for shell space and describe techniques that use thick shells and novel aspects of such mapping for volumetric detail, since the "ceiling" case above is a 2D mapping that relies on rendering algorithms to give the appearance of volumetric detail. At the end of the section, we provide a relative comparison of techniques.

4.1. Mapping Volumetric Detail on Meshes

In traditional GPU-accelerated rasterization, 2D texture coordinates are linearly interpolated over triangle surfaces, and the mapping between texture and object spaces is affine. When using quadrilaterals, patches, or quadrilateral-based subdivision surfaces as primitives, the mapping between the spaces needs to be more complex, such as bilinear or projective, with the former being more natural for these spaces (Figure 6). These mappings can be extended to 3D using the volume equivalent of the surface primitives: quadrilaterals extruded to hexahedra and triangles extruded to prisms. An excellent review for 2D mapping approaches is given by Heckbert [Heckbert 1989].

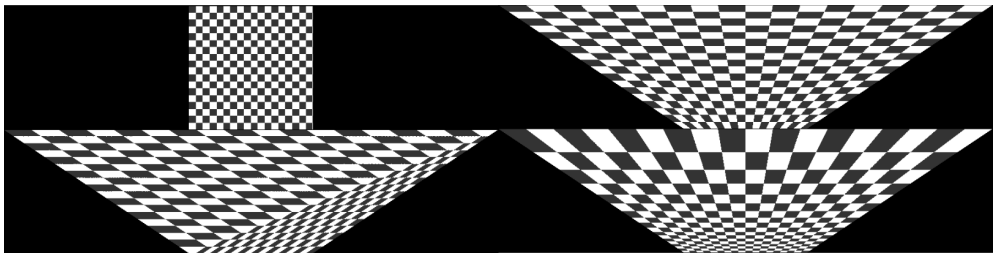


Figure 6. A textured quad rendered as two triangles (top left). Deforming two upper vertices: while the shape is a quadrilateral, the separate triangles can clearly be seen as a result of the affine texture mapping (bottom left). Bilinear mapping (top right) and projective mapping (bottom right) do not exhibit this problem.

The surface $S(u,v)$ is typically discretized to a triangle mesh. Let us then denote as T_{123}, a triangle specified at coordinates $T_i = (u_i, v_i), i \in [1,3]$ and its barycentric coordinates as $\vec{b} = (s,t,q)$ where $s+t+q = 1$. A discretized shell space, therefore, replaces triangles with their corresponding prisms:

$$P(\mathrm{T}_{123}) = \int_{w=0}^{1} G(\mathrm{T}_{123}, w)\, dw$$

Triangles become prisms with bilinear patches for fins, as the resulting quadrilateral faces are generally not planar. A point $\vec{q}(\vec{b}, w)$ inside a prism can be obtained as

$$\vec{q}(\vec{b}, w) = \sum_{i=1}^{3} G(b_i T_i, w). \tag{1}$$

This is the equivalent of linearly interpolating by w the points evaluated at $\vec{b}$ on the bottom and top triangles of the prism.

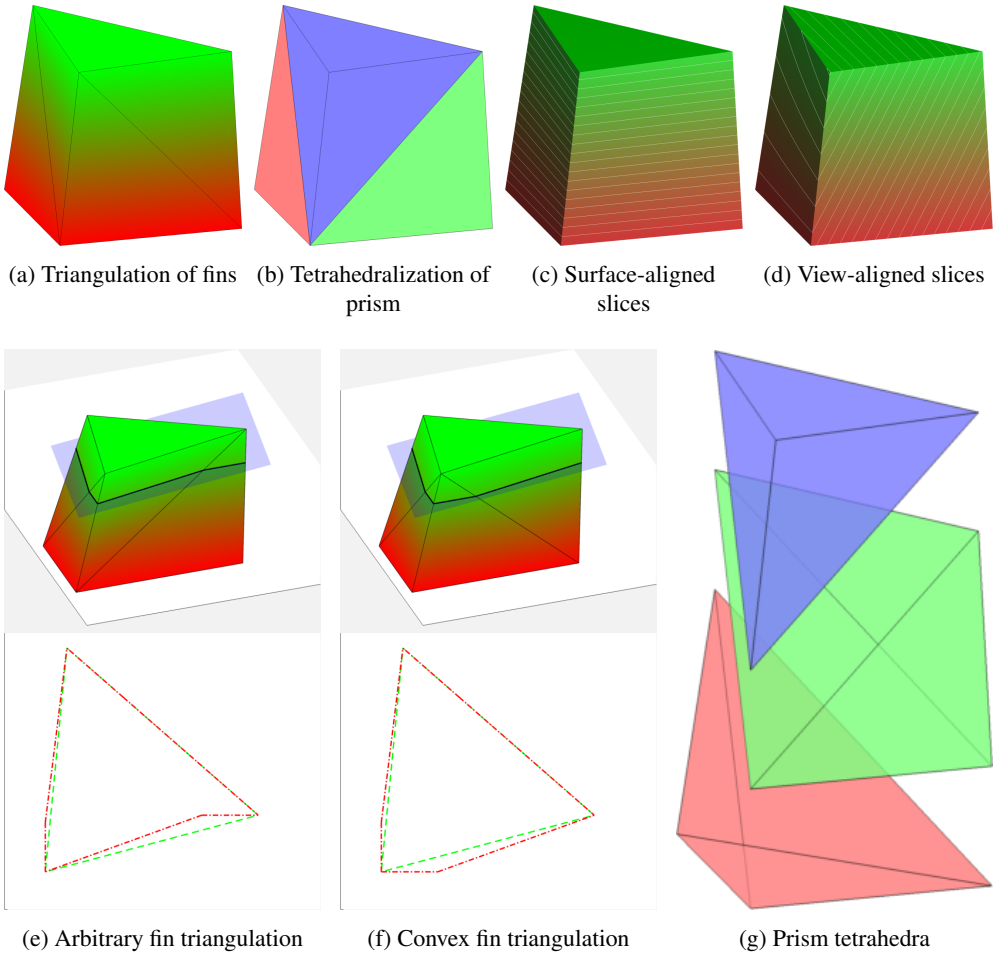
Quadrilaterals and patches are similarly extended in 3D, and the mapping becomes trilinear. More details on the mappings can be found in Appendix C.

Most shell-based techniques represent these volumes explicitly using geometric primitives. But since most real-time rasterizers use triangles as rendering primitives, the prism/hexahedral fins need to be approximated with triangles (Figure 7). Depending on the diagonal used for triangulation of each fin, a bilinear patch might not necessarily be fully contained inside the coarse triangular approximation (Figure 7(e)). For a watertight shell mesh volume, the triangulations of fins shared by adjacent prisms need to match, although, as seen in Section 4.2, that is not always necessary.

4.2. Shell mapping

[Kajiya and Kay 1989] first described volumetric textures for use with fur. They mapped *texels* (3D volumes) on bilinear patches, and inside each texel they stored densities and reflectance properties per-voxel. This representation was later extended by [Neyret 1995] by using a multiscale representation for texel data and was later improved by [Meyer and Neyret 1998] to run at interactive frame rates. A texel volume is mapped on the volume defined by the extrusion of a bilinear patch towards the surface normals defined at the bilinear patch corners.

[Peng et al. 2004] improve on the parameterization of the shell space and avoid self-intersections of the offset suface (AvDf). They use multiple layers with *slabs*, and allow the shells to be interior, exterior, or enveloping (with layers over and below the surface). Each slab is a prism whose vertices are extruded from the base surface by a specified thickness h towards vectors called *line directors*. Directors are obtained as the gradients of a modified distance function (L_p*-averaged distance function*) that is continuous and has a gradient field with similar properties to the Euclidean distance



(a) Triangulation of fins (b) Tetrahedralization of prism (c) Surface-aligned slices (d) View-aligned slices

(e) Arbitrary fin triangulation (f) Convex fin triangulation (g) Prism tetrahedra

Figure 7. Shell geometry representations for rendering. Bottom parts of subfigures 7e and 7f show the horizontal slice of the prism from a top view using green color for the original prism volume silhouette and red for the fin approximation with triangles.

gradient field.

The extrusion amount towards the director lines, as well as the number of layers, are stored per-vertex. While initially the parameterization is slow to calculate, it can be updated efficiently for small deformations.

[Wang et al. 2004] use GnDm to apply volumetric detail on triangle meshes. They map the (extruded to 3D) texture space of a triangle mesh to the corresponding shell space, so uniform triangular prisms in texture space are mapped to shell-space triangular prisms. The shell-space bilinear prism fins are approximated by two triangles per fin (Figure 7(a)).

At the same time, [Hirche et al. 2004] use a similar approach for displacement mapping. They assume a triangle mesh as the *base surface* and define the *offset sur-*

face as the mesh extruded towards the direction of the surface normals. Prisms are further partitioned into tetrahedra (three tetrahedra per prism, Figures 7(b), 7(g)) to accelerate rendering, and they use vertex-ordering information to construct tetrahedra such that neighboring prisms match continuously without any T-junctions. The mapping is a straightforward barycentric mapping between corresponding shell-space and texture-space tetrahedra. The piecewise-linear nature of the mapping results in C^0 continuity between adjacent tetrahedra or prisms.

[Porumbescu et al. 2005] define *shell maps* (Sm) as bijective maps between texture space and shell space and use this concept to map volumetric detail. Similar to [Hirche et al. 2004], they extrude a triangle mesh and map the resulting shell to 3D texture space and partition each prism to tetrahedra. They additionally avoid self-intersections in the offset surface by reducing the extrusion distance of the intersecting primitives. In order to avoid T-junctions from adjacent prism tetrahedra, they use a floodfill-like algorithm. The mapping is used to apply volumetric mesostructure in any form (procedural, texture, geometry) to surfaces.

A similar technique was concurrently developed by [Dufort et al. 2005] to render semi-transparent volumetric detail at interactive frame rates (STSm). They use a triangle mesh for the base surface and generate an offset surface by extruding the mesh towards the direction of the normals by a fixed amount. The prism is split into three tetrahedra, similar to [Hirche et al. 2004] and [Porumbescu et al. 2005].

[Jeschke et al. 2007] improve the smoothness of the shell mapping while maintaining interactive frame rates (SSm). As tetrahedra-based approaches [Hirche et al. 2004; Porumbescu et al. 2005; Dufort et al. 2005; Ritsche 2006] result in artifacts inside a prism due to the piece-wise linear nature of the tetrahedral mapping, they avoid such artifacts by triangulating the prisms so that the fins are convex (Figure 7(f)). They also introduce a curved mapping (CSm) to improve smoothness across prisms by maintaining tangent continuity at prism boundaries: For each w-coordinate of (u,v,w) in a texture space prism, a Coons patch is defined for its corresponding world space triangle $G(T_{123},w)$. To avoid the patch protruding from the geometry, w is compressed to a heuristic value of $w' = \frac{w}{2} + \frac{1}{4}$. Even though the distortion when using the curved mapping is minimised and controllable, the technique adds a performance overhead while the space warping alters the perceived macrostructure surface, which is not desirable on model surfaces on which such mesostructure is partially covering.

Other techniques that use the basic shell formulation (extrusion of triangular mesh towards normals) to map volumetric mesostructure have been developed by [Ritsche 2006] and [Decaudin and Neyret 2009]. Shell construction methods are grouped in Figure 8.

[Zhou et al. 2006] reduce distortions caused by naive parameterizations by minimizing a stretch metric (StreMe) on the shells' tetrahedra, using the fact that the

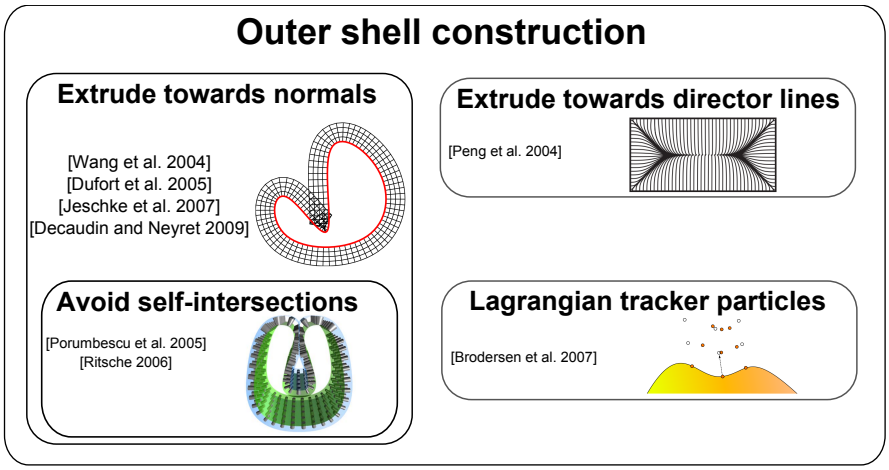


Figure 8. Outer shell construction. Figures from [Porumbescu et al. 2005; Peng et al. 2004; Brodersen et al. 2007]

Jacobian $\mathbf{J}$ of the object-space-to-texture-space shell mapping function G^{-1} is constant over each tetrahedron due to the piecewise linear nature of the mapping. The eigenvalues of the Cauchy deformation tensor $\mathbf{J}^\top\mathbf{J}$ can be used to compute the root-mean-square stretch. The mapping is optimised using iterative minimization of the stretch metric, by performing a random line search on the (u,v)-plane, resulting in modified (u,v)-coordinates for the offset surface vertices.

[Brodersen et al. 2007] use *Lagrangian tracker particles* (LaTraP) for detail space parameterization. Such particles can be distributed at unique (u,v,w)-coordinates over a patch volume and can be optimised to reduce various types of distortions. A very simple distribution is a *surface conforming parameterization*, in which particles are propagated along the gradient field direction until the required offset is reached, but it is prone to distortions on areas of high curvature. Another distribution is a *reduced distortion level set parameterization*, in which a number of particle levels is used, each with unique parameterization and particle density, while users can specify offset directions. A different form of parameterization uses a *spline advection* scheme, in which particles are propagated along a spline curve originating at the center of each patch.

In the above schemes, the particles form a 3D lattice composed of multiple layers of regular 2D grids, each sized differently. They suggest two mapping algorithms, one targeted for interactive performance (mapping a geometric texture to an implicit surface), and one for higher-quality, but non-interactive (mapping an implicit surface texture to an implicit surface). The interactive algorithm obtains the target coordinates by applying trilinear interpolation on this semi-regular grid. Given a point (u,v,w), the (u,v) points on the nearest w layers (above and below) are calculated via bilinear interpolation, and the final value is obtained by linearly interpolating these two points.

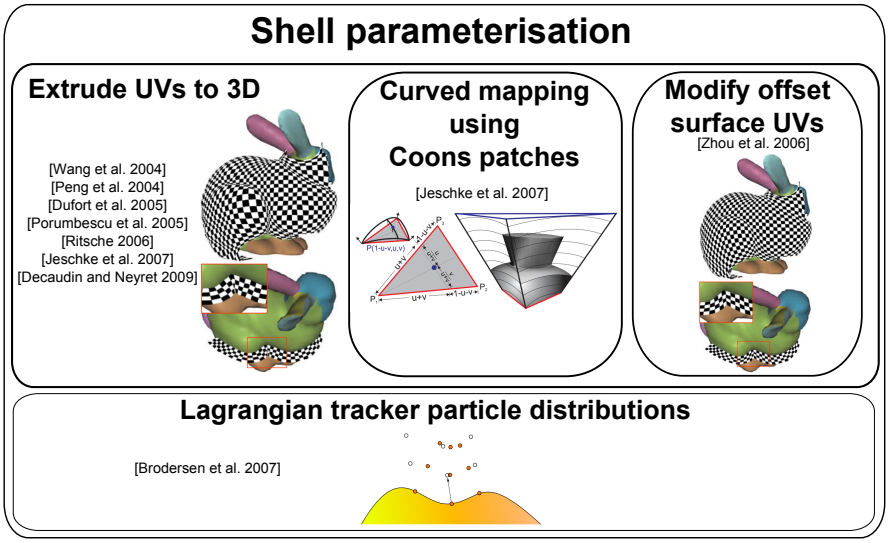


Figure 9. Shell parameterization. Figures from [Zhou et al. 2006; Jeschke et al. 2007; Brodersen et al. 2007]

From the above we can observe that the more complex parameterizations which produce better results [Peng et al. 2004; Zhou et al. 2006; Brodersen et al. 2007] are too slow to calculate in real-time. Shell parameterizations are grouped in Figure 9.

4.3. Summary

Table 3 summarizes work on offset surface construction and shell parameterization. For abbreviation, we will refer to simple extrusion towards normals as ETN, self-intersection checks as ETNSI, the Lagrangian tracker particle grid as LaTraP, the tetrahedron stretch reduction method as StreMe, and finally the L_p-averaged distance function as AvDf. Similar to Table 2, the ratings are based on a consensus among the authors and are intended to serve as useful, high-level comparison measures.

Computation time. ETN is the fastest method since it requires minimal computations; it is mainly used by interactive volumetric texture-mapping methods. Checking for self-intersections (ETNSI) and adjusting heights requires extra processing but can still be interactive, while solving non-linear optimizations (LaTraP, StreMe, AvDf) requires significantly more processing power and these techniques are mainly used by offline methods or as a preprocessing step.

Memory. ETN does not require any memory, while ETNSI requires per-vertex height adjustments to avoid self-intersections. StreMe stores (u,v)-coordinates per-vertex, while LaTraP requires storing grids per patch, per shell layer, and AvDf requires storing 3D textures in addition to per-vertex information.

Who	Offset generation and parameterization technique	Computation time	Memory	Mapping distortion	Mapping control
[Wang et al. 2004] (**VuDm**) [Dufort et al. 2005] (**STSm**) [Jeschke et al. 2007] (**SSm/CSm**) [Decaudin and Neyret 2009] (**VolB**)	**ETN**	$<$ msec	0	★☆☆	★☆☆
[Porumbescu et al. 2005] (**Sm**) [Ritsche 2006] (**RTSS**)	**ETNSI**	$<$ sec	20 KB	★★☆	★☆☆
[Brodersen et al. 2007] (**LaTraP**)	**LaTraP**	$>$ sec	48 MB	★★★	★★★
[Zhou et al. 2006] (**StreMe**)	**StreMe**	$>$ sec	40 KB	★★★	★☆☆
[Peng et al. 2004] (**AvDf**)	**AvDf**	$>$ sec	64 MB	★★★	★★☆

Table 3. Comparison of shell-based techniques with regard to shell construction and parameterization. The numbers are estimated for a mesh of 5K vertices and grid resolution of 1K$\times$1K$\times$4 (LaTraP,AvDf). For more details about the abbreviated methods, categories, and ratings, refer to Section 4.3.

Mapping distortion. ETN results in the worst mapping, as distortion artifacts are amplified at areas of high curvature and self-intersections occur at concave areas. ETNSI just corrects the above self-intersections. LaTraP, StreMe, and AvDf minimize distortions of the offset surface mapping.

Mapping control. ETN, ETNSI, and StreMe do not provide control over the offset surface mapping. AvDf allows control over the shape of the gradient field lines, while LaTraP offers a variety of distortion metrics for the mapping.

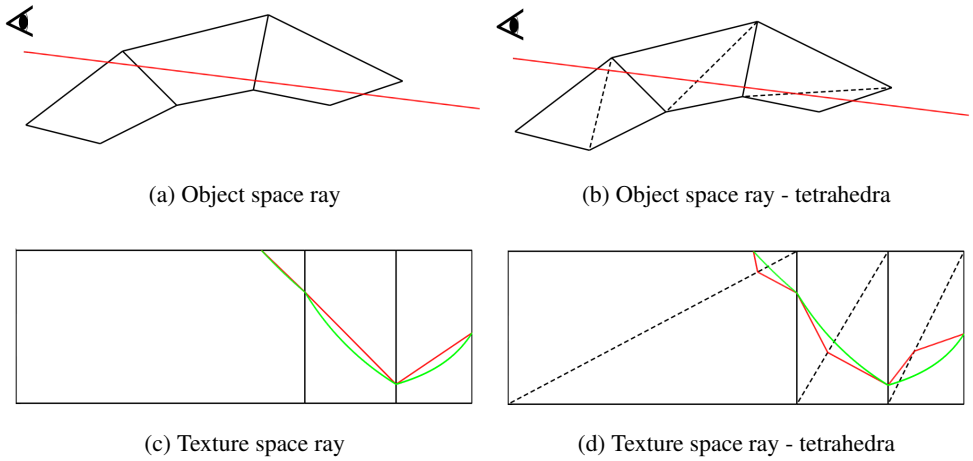
5. Rendering

Real-time rendering techniques that can be used to display volumetric mesostructure, using such storage representations or mappings from the previous sections, can be split into three groups: *raycasting*, *precomputed texture function lookups* and *slice-based rendering*. Below, we review techniques in terms of their rendering contributions, and we also note issues related to rendering, such as distortions caused by rendering or deformations and lack of support for silhouettes.

5.1. Raycasting

Many volumetric texturing techniques render the coarse geometry (base and/or offset surfaces) and cast rays to it. The rays traverse the mesostructure shell space until they intersect with the shell again, or until they get absorbed by the contained mesostructure. Methods calculate either the light transport along the ray (for translucency) or the first intersection with opaque mesostructure. Rays are traversed in either object or texture space, each space having its own traversal characteristics.

Object space traversal requires the evaluation of texture coordinates at each step for sampling the mesostructure. Texture coordinates are typically interpolated from



(a) Object space ray

(b) Object space ray - tetrahedra

(c) Texture space ray

(d) Texture space ray - tetrahedra

Figure 10. Side view of a shell in object space (top) and the corresponding view in texture space (bottom). The ray passes through three extruded faces. The correct traversal path in texture space is shown in green. The piecewise linear approximation, obtained from interpolating entry to exit points, is shown in red. The right side shows the errors of the ray path approximation if we subdivide each extruded face to tetrahedra, shown here as triangles.

values specified on the corners of the primitive they are in. When rendering shells as extruded triangulated prisms, texture coordinates are interpolated on the triangle faces but need to be calculated when a ray is inside a prism. So, at each raycasting step, this requires the inversion of Equation (1) with $\vec{q}(\vec{b},w)$ known and $\vec{b}$, w as unknowns:

$$(\vec{b},w) = \vec{F}(x,y,z) = \vec{q}^{-1}(\vec{b},w)$$

Texture space traversal does not need such a transformation per-step, as it can sample mesostructure directly. But because the transformation from object space to texture space is not globally affine, rays in object space become curved in texture space and vice versa. So, a straight line traversal in texture space is frequently an approximation of the object space ray traversal and can result in visual artifacts (see Figure 10). Techniques have attempted to curve texture-space rays in a variety of ways and contexts: using "deflectors" [Kurzion and Yagel 1995], using numerical solution [Neyret 1996], barycentric correspondence [Jeschke et al. 2007], tangent/normal map lookups [Chen and Chang 2008], or "ray direction" textures [Na and Jung 2008].

[Porumbescu et al. 2005] compute the world space ray intersections with either the base or the offset surface. The intersecting tetrahedron's entry and exit point are computed, and the ray segment is marched in world space. At each step, points are transformed in texture space using point-in-tetrahedron queries and barycentric coordinates, in order to calculate densities and ray-surface intersections. Because ray

marching is performed in world space, they actually trace a curved ray in texture space, which does not introduce texture distortions due to space warping for small step sizes. Each object in texture space has a predefined step size, resulting in adaptive sampling. Tetrahedra store links to adjacent tetrahedra, so that if a ray exits through one of the sides, ray marching continues. The implementation of this technique was non-interactive at the time of its development, but it is possible to implement in GPUs to run at interactive frame rates in today's hardware due to increased programmability and performance.

[Dufort et al. 2005] initially sort the shell tetrahedra using the "scanning exact meshed polyhedra visibility ordering" (SXMPVO) algorithm by [Cook et al. 2004] when rendering semi-transparent textures. For each rendered tetrahedron, the vertex shader computes the intersection of the ray originating from each vertex towards the direction of the view ray with one of planes formed using the tetrahedron's faces. The intersection point is transformed to texture space, and the coordinates of both spaces are interpolated as they are passed to the pixel shader. Ray marching is performed on the texture view ray and color is accumulated based on the opacity of the ray-marched samples.

[Donnelly 2005] presents a GPU-accelerated version of sphere tracing by [Hart 1996], applied to rendering displacement maps instead of implicit surfaces (DfDm). It is similar to normal mapping techniques, but uses a 3D texture storing a distance field in texture space. The view vector is transformed in texture space and sphere tracing is used to traverse the distance field and compute the hit point.

While the evaluation is fast, there are a number of drawbacks, such as deformation distortions, storage cost, generation cost, and lack of silhouettes. While the algorithm performs well on planar surfaces, artifacts will appear on curved surfaces. This is a result of the precomputed distance data in the 3D texture space. Since the volume is warped because of surface curvature, the precomputed closest distances become invalid (see Appendix B, Figure 11). Even if the distance field data are calculated while mapped uniquely on the surface, any deformation results in artifacts. The distance field generation cost is also prohibitive for dynamic data generation, resulting in inability to represent procedurally defined surfaces; that is, if the distance field cannot be represented procedurally as well. While this technique has several drawbacks, it is a simple but effective way of accelerating raycasting and was used in later shell-mapping techniques [Ritsche 2006; Jeschke et al. 2007].

[Ritsche 2006], in *real-time shell-space rendering* (RTSS), uses the same ray-plane intersection scheme as STSms. For raymarching, a fixed number of iterations is used but the distance traversed for each step is based on values obtained from a mapped 3D distance field sampled at each step.

As mentioned in Section 4.2, all tetrahedra-based shell mapping techniques [Hirche et al. 2004; Porumbescu et al. 2005; Dufort et al. 2005; Ritsche 2006] ex-

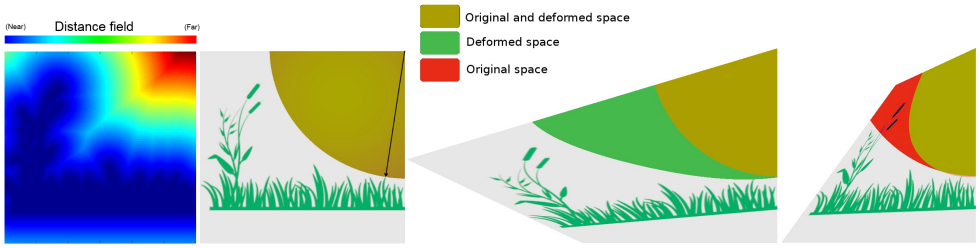


Figure 11. Distance field of a texture (left) and sampling on a quad in original (middle left) and deformed (middle right, right) object spaces at the top-right corner. The region within the radius of the sampled safe distance, valid for the undeformed state of the texture (original space) is shown in red. The same region, deformed along with the texture, is shown in green. The overlapping part of both regions is shown in brown. Notice how the sampled distances in the deformed spaces can be either too conservative or too large if used for object-space ray traversal acceleration.

hibit artifacts due to the piecewise-linear nature of the mapping. [Jeschke et al. 2007] avoid such artifacts by iterative correction of the ray path. They first triangulate the prism fins so that the prisms are convex . They observe that they can partially invert the function in Equation (1), which maps texture space to object space by using the *barycentric correspondence* of triangles at a fixed w coordinate:

Given an offset w, the barycentric coordinates $\vec{b}$ of a point $\vec{q}(\vec{b},w)$ on a triangle $G(T_{123},w)$ correspond exactly to the barycentric coordinates of T_{123}, allowing for an easy transformation between object and texture space.

The view-ray segment end is calculated by computing the intersection with all the prism triangles, and the w values are computed for the entry and exit world space points using a bisectional search. Sampling along the ray is performed by iteratively correcting the linearly interpolated ray in texture space; at each step, the sampled point is transformed to world space, adjusted to the correct position and then transformed back while also adjusting the texture space sampling direction. This adjustment occurs a few times until the error is low enough. The process is repeated until an intersection is found or the exit point is passed. They also use 3D distance fields to accelerate raymarching.

In contrast to shell-mapping techniques, [Policarpo and Oliveira 2006] approached volumetric mesostructure using a layered relief-mapping approach. Intersection testing is done similar to relief maps, but in this case it is performed in parallel for each layer, and, in the end, only the closest intersection is kept. As in relief maps, the technique supports silhouettes. Correct silhouettes are obtained by precomputation of two per-vertex coefficients, a and b, that represent a quadric surface ($z = ax^2 + by^2$) which locally approximates the underlying geometry. Since correct silhouette rendering requires calculations based on the surface geometry, it results in an additional performance cost on animated or deformable surfaces. Also, if the rays have to travel

far in the texture, the quadric representation becomes a poor approximation and will result in artifacts.

5.2. Precomputed Texture Function Lookups

Instead of marching along rays, techniques using high-dimensional texture functions do view-dependent texture lookups, effectively trading off runtime calculations with storage requirements.

[Wang et al. 2003] calculate the curvature along the tangent-space view direction as a function of the view direction, the principal curvatures, and the surface normal. They then use the curvature, the view direction, and the texture coordinate to sample the function. Pixels for which the sampled distance is -1 (special value for specifying no intersection of ray with geometry), are outside of the silhouette and are discarded.

[Wang et al. 2004] apply a two-pass approach using graphics hardware. In the first pass, vertices are projected towards the viewing direction and are tested for intersection with the planes containing the prism backfaces. Distances and texture coordinates of plane hit points are rasterized and used in the second pass as vertex attributes. The interpolated values in the pixel shader, along with the computed texture space view direction, are used to sample the GnDm and obtain the mesostructure distance. If there is an intersection, the 3D texture coordinate is computed so shading can be performed.

[Wang et al. 2005] use a depth-peeling approach for silhouette determination: They render front and back faces in each peeling pass, sampling from the MsDf and comparing distances to identify if there is any mesostructure along the resulting ray segments.

All of the above techniques exhibit artifacts under macrostructure deformation, since the precalculated visibility data are computed for a single mesh configuration.

5.3. Slice-based Rendering

A shell-rendering alternative to the previously described techniques is to partition each extruded volume to aligned slices and render them; this was first introduced by [Meyer and Neyret 1998].

[Peng et al. 2004] use a slice-based direct isosurface rendering method implemented in two passes, where the slices are perpendicular to the surface normal (Figure 7(c)). In the first pass, for each quadrilateral face, they render the layers of the face as stacked quadrilaterals extending towards the line director vectors, and output the depth and 3D coordinates of the fragments nearest to the isosurface. In the second pass, they smoothly interpolate the coordinates and obtain normals by transforming the stored gradient from texture space to the space where shading is applied.

More recently, *volumetric billboards* (VolB) were introduced by [Decaudin and Neyret 2009] as a rendering technique used for volumetric mesostructure rendering

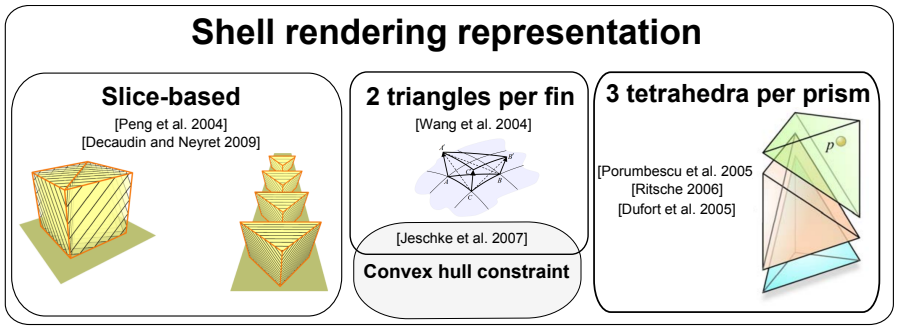


Figure 12. Shell rendering representations. Figures from [Porumbescu et al. 2005; Decaudin and Neyret 2009; Wang et al. 2004]

with opaque as well as semi-transparent content. It requires a 3D volume representation of colors, opacities, and normals and uses *cells*, which are prisms extruded from a macrostructure surface. The scene is partitioned in *slabs*, rectangular volumes orthogonal to and along the camera's view direction, and each slab is assigned only the cells that intersect it. For each slab, the assigned cells are rendered, and the geometry shader generates the polygon that defines the prism-plane intersection surface. Slabs are processed in back-to-front order for correct blending. The shell is rendered using a GPU-accelerated slice rendering scheme, generating view space slices for each cell in the geometry shader (Figure 7(d)).

The technique is real-time, produces good results, and can handle silhouettes, mesh deformation, and mesostructure transparency, but its main problem is the number of slices it needs for close-up detail. The bottleneck of the technique is the fill-rate and, in particular, the rendering in back-to-front order. Another issue is the normal sampling, since precomputed normals cannot represent thin geometry in volume data very well. Artifacts are also expected from a small number of slices when rendering volumes containing regular as opposed to fuzzy structures, while the required number of slices for prisms extruded from a relatively dense triangle mesh could again cause more performance issues. A potential performance bottleneck is the assignment of cells to slabs, since there is no mention of how this would scale to more than a few thousand cells.

A grouping of shell-based rendering techniques based on the geometric representation of the shells is shown in Figure 12.

6. Rendering Artifacts, Limitations and Performance

Interactive rendering techniques often need to sacrifice quality or features (Figures 13 and 14). In this section, we describe visual artifacts and limitations exhibited by the reviewed rendering techniques. Table 4 summarizes this information. We conclude the section with a short discussion about the difficulty of a fair performance comparison.

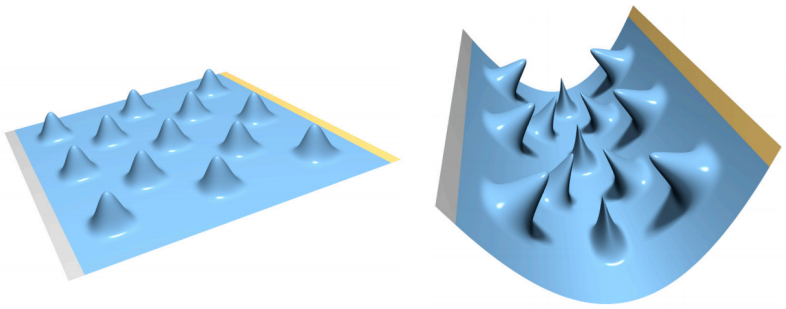
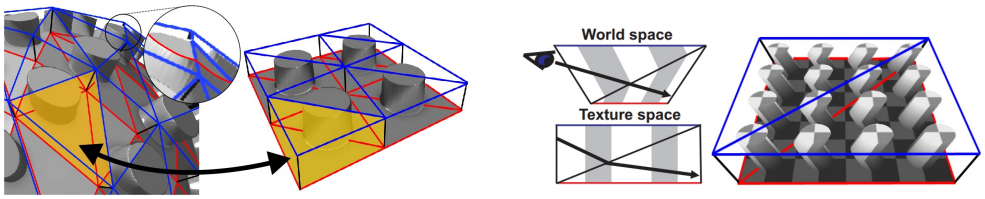


Figure 13. A plane using displacement mapping for bumps (left). When the plane bends (right), the volumetric space near the bend is compressed, resulting in the nearby bumps getting distorted. Figure from [Botsch and Sorkine 2008].



(a) Prism artifacts. Notice how the top boundary curve of the zoomed cylinder abruptly changes at the prism boundary.

(b) Tetrahedra artifacts. Notice how the cylinders bend abruptly inside the prisms.

Figure 14. Examples of shell-mapping artifacts. Figures (modified) from [Jeschke et al. 2007].

Below, we use the abbreviated forms of the reviewed techniques: VuDm is [Wang et al. 2003], AvDf is [Peng et al. 2004], GnDm is [Wang et al. 2004], MsDf is [Wang et al. 2005], STSm is [Dufort et al. 2005], DfDm is [Donnelly 2005], RTSS is [Ritsche 2006], MLRm is [Policarpo and Oliveira 2006], SSm/CSm is [Jeschke et al. 2007], and VolB is [Decaudin and Neyret 2009].

6.1. Silhouettes

Silhouette rendering is supported by all techniques that render and raymarch the extruded shells. Techniques that render only the original mesh surface have a greater difficulty rendering silhouettes, since they have no information about how the mesh changes along the ray direction. Curvature precalculation (VuDm) and local surface approximation as quadric (MLRm) have notable performance costs, thus excluding common mesh animations. Silhouette determination with MsDf is performed using depth peeling, which does not scale well in the number of layers in terms of performance. Simple texture-space ray traversal, like the one used in DfDm, does not allow rays to escape the texture space volume and thus cannot display silhouettes.

Artifacts and limitations			**VuDm**	**AvDf**	**GnDm**	**MsDf**	**STSm**	**DfDm**	**RTSS**	**MLRm**	**SSm**	**CSm**	**VolB**
Silhouettes						✗		✗		✗[1]			
Dynamic mesostructure			✗		✗	✗		✗	✗	✗	✗	✗	
Transparency			✗	✗	✗	✗		✗	✗	✗	✗	✗	
Shell self-intersections					✗		✗				✗	✗	✗
Ray path	Curved	Mapping	✗		✗	✗	✗		✗				
		Deformation	✗		✗	✗	✗	✗	✗	✗[1]	✗	✗	
	PWL	Prism fins			✗		✗		✗		✗	✗[2]	✗
		Tetrahedra					✗		✗				

Table 4. Artifacts and limitations of interactive mesostructure rendering techniques, ✗ notes the existence of an artifact or lack of support for a feature. Abbreviations are linked to techniques in Table 1 and Section 6.

6.2. Dynamic Mesostructure

Support for dynamic mesostructure excludes techniques that rely on non-interactive precomputations based on the mesostructure content. As such, techniques that rely on distance fields are excluded (DfDm) as distance field calculation is currently quite performance intensive for large datasets, while techniques that use them as an optional acceleration structure become significantly slower without them (RTSS, SSm, CSm). Procedurally defined distance fields are possible [Quilez 2008], but are very difficult to generate for arbitrary detail. High-dimensional texture functions are also excluded, since they rely on generation of the samples and their subsequent compression, both operations being non-interactive and done as a preprocess (VuDm, GnDm, MsDf). MLRms are also excluded, since the conversion from a volumetric representation to depth layers is part of preprocessing and quite performance intensive.

6.3. Transparency

While transparency is a material property, it imposes significant requirements on algorithms for rendering shape, such as primitive sorting and visibility integration, so that, in most cases, it is simply unsupported. Techniques that were designed to explicitly support transparency are STSm and VolB. This type of transparency (material) is not to be confused with transparency as a result of integrating visibility of fine geometry when sampling.

[1] Artifacts are mutually exclusive

[2] Assuming Coons patches as the limit base surface results in artifacts when using partially covering volumetric mesostructure on a different base surface

6.4. Shell Self-intersections

As mentioned in Section 4.2, the naive way of creating an offset surface by extruding the base surface by a fixed amount (GnDm, STSm, SSm, CSm, VolB) can result in shell self-intersections in concave areas which, in turn, results in rendering artifacts.

6.5. Ray Path

The most common source of errors is the ray traversal through the mapped volume data. Below, we explain and name the reasons for these errors.

6.5.1. Curved, mapping

As explained in Section 5.1 and shown in Figure 10, straight rays in object space become curved in texture space and vice-versa.

VuDm does not take into account the change of curvature on the surface while MsDf uses a depth-peeling approach that is limited in the number of layers it can process before the rendering speed drops significantly. GnDm exhibits fewer visual artifacts than VuDm by calculating a better approximation of the ray path in texture space, but they still do not account for the ray curving.

All techniques that use tetrahedra (STSm, RTSS) are subject to these errors, since they approximate the curved ray path as a coarse, piecewise-linear curve.

6.5.2. Curved, deformation

Similar to the above errors, curved ray paths in texture space that correspond to straight ray paths in the object space of a mesh in one pose, are no longer correct if the mesh undergoes a non-affine deformation to a different pose.

All techniques that exhibit artifacts for static objects naturally continue to exhibit such artifacts when the surface is deforming (VuDm, GnDm, MsDf, STSm, RTSS).

MLRms can approximate the surface locally as a quadric and calculate a curved ray path in texture space. However, if the surface deforms, the approximation has to be recomputed, resulting in a greater computational cost.

Techniques that use distance fields for acceleration of ray casting (DfDm, RTSS, SSm, CSm) are subject to artifacts due to the invalidation of the distance field under deformation, as discussed in Appendix B.

6.5.3. Piecewise-linear, prism fins

In most interactive volumetric texture rendering techniques, the mesh surface is discretized to triangles, and the 2D piecewise-linear (PWL) parameterization of the base mesh is extended to 3D by assigning texture coordinates to the offset surface and linearly interpolating the space inside. Rays in shell space are curves in texture space and, due to the piecewise-linear nature of the mapping, these curves are C^0-continuous at the (bilinear) fins. As shown in Figure 10, linear interpolation of entry/exit texture coordinates for the space inside each prism results in continuity artifacts at the prism

fins. Such artifacts become worse if the fin is approximated by triangles. The continuity artifacts are observed in most techniques that use shells (GnDm, STSm, RTSS, SSm, VolB).

6.5.4. Piecewise-linear, tetrahedra

As shown in Figure 10, techniques that discretize prisms to three tetrahedra (STSm, RTSS) introduce different rendering artifacts compared to prisms with triangulated fins: the approximation is better but there are more derivative discontinuities in texture space.

6.6. Performance

During the last decade, the power and programmability of GPUs have significantly increased, so a fair performance comparison of techniques in the way they were developed is very difficult. Techniques that were previously non-interactive and implemented in the CPU, can be now suitable for a GPU implementation (e.g., [Porumbescu et al. 2005]). Another example is shell geometry generation and rendering which, with modern hardware, can be modified to use geometry shaders, thus changing the performance characteristics of techniques that use them (GnDm, STSm, SSm, CSm, VolB).

In Table 5, we provide a performance comparison of texture-mapped volumetric detail rendering methods that share characteristics with many of the reviewed techniques. We used a system equipped with a GeForce GTX Titan, rendering a close-up of two triangles extruded to a prism at a resolution of 1920×1200 in order to avoid a bottleneck in the geometry shader where we chose to construct the prism. No optimizations were applied. For triangle and tetrahedra shells, raycasting step size was set at voxel resolution. For slice-based rendering, 40 slices were used, each subdivided to 5 in the geometry shader.

Extra features	**Triangle shell**	**Tetrahedra shell**	**View-space slice-based**	**Multi-layer relief mapping**
None	2.85 (6.89)	(**STSm**) 5 (12.5)	(**VolB**) 27.02 (33.33)	(**MLRm**) 1.05 (1.10)
Distance field	1.85 (3.24)	3.57 (5.71)	N/A	N/A
Curved ray path	28.57 (66.66)	(**Sm**) 100 (250)	N/A	N/A
Distance field + Curved ray path	(**SSm,CSm**) 22.22 (45.45)	(**RTSS**) 25 (50)	N/A	N/A

Table 5. Performance comparison of shell-rendering techniques, rendering a $256 \times 256 \times 64$ volumetric texture mapped in a shell extruded from two triangles. Times are in milliseconds. Times in parentheses are for the same volumetric texture at double resolution ($512 \times 512 \times 128$). The techniques in parentheses show the shell rendering method and the extra features that they use.

7. Summary and Conclusions

Volumetric mesostructure rendering using texture mapping methods is a difficult problem. Current methods trade-off quality and flexibility to be interactive, and no single method is artifact-free, even at the expense of performance. Texture mapping methods were chosen for their support of deformable surfaces and, as such, we focused on techniques that contribute to one of the three core parts of such a rendering pipeline: storage, mapping, and rendering.

We described storage methods such as procedural definitions, dense, hierarchical and compressed texture maps and meshes. Mesostructure data are mapped either on a thick shell over the surface or directly on the surface. Real-time rendering methods include slice-based volume rendering for shells, rendering the shell geometry and casting rays through it, or using the view rays and the location on the base surface or shell to sample a high-dimensional texture map.

With regard to storage, dense volume textures do not scale well to high-resolution data. Procedural methods are still non-intuitive for modeling of volumetric mesostructure, while the evaluation cost can be prohibitive for real-time applications. Geometry as a mesostructure representation has not been generally used due to its numerous disadvantages. High-dimensional texture functions have high storage and precomputation costs and do not scale well with high frequency/resolution detail.

For rendering, slice-based methods can be effective when a low number of slices is required or the detail is stochastic, but fill rate becomes a problem for highly detailed representation of non-stochastic mesostructure. Raycasting is a well-studied rendering technique, but when naively applied to warped 3D domains (such as traversal of texture space using world-space rays) distortions are introduced. Additionally, raycasting acceleration methods—such as distance fields—that rely on offline precomputations dependent on macrostructure, result in artifacts if the latter is deformed.

7.1. Open Problems

Although volumetric texture-mapping research has been ongoing for decades, there are still a lot of open problems. Addressing these problems will allow for better quality, interactive volumetric mesostructure rendering on deformable surfaces.

Hierarchical storage for volumetric textures. Most of the reviewed techniques are either agnostic to the representation of detail, or use custom representations, or use some simple but non-scalable forms such as geometry or dense textures. With recent developments in out-of-core rendering (Section 3.2.2), where highly detailed volumes are stored and accessed efficiently using hierarchies, we expect to see these concepts applied to volumetric textures.

Streaming for volumetric textures. The growing demand for higher detail complexity results in storage requirements increasing much faster than the availability of GPU

memory and, more importantly, bandwidth between disk, application, and GPU. This has led to increased research in streaming for out-of-core techniques. As in the case of 3D mesostructure, where the storage requirements are usually prohibitive for large amounts of unique detail, we can expect adoption of streaming for such volumetric texturing techniques.

Dynamic, low distortion 3D parameterization for deformable surfaces. Existing low-distortion 3D parameterizations are performance intensive even for calculating single frames. Since a major benefit of volumetric texture mapping is its application on deforming surfaces, we expect 3D parameterization methods to exploit the spatially coherent nature of the deformation in order to dynamically update the parameterization. Content-aware techniques, such as [Koniaris et al. 2013], can be developed for 3D parameterizations in order to exploit the non-homogeneity of the volumetric materials.

Efficient ray traversal in 3D-parameterised, deformable volumes. Ray traversal has been a major subject of research in rendering of static volumes. Given the visual artifacts existent in many of the current ray traversal methods for volumetric textures, we expect researchers to try and adapt recent ray traversal research to deformed volumetric domains.

A. Volumetric Space Normals

Given a 3D density field $D(u,v,w)$, normals can be obtained as follows:

$$\hat{N}(u,v,w) = -\frac{\nabla D(u,v,w)}{\|\nabla D(u,v,w)\|}$$

If the density field is given as a volumetric texture mapped to the surface's shell space, the obtained normals need to be transformed according to the shell space transformation. Using the generalization of the Taylor series, $G(\vec{q})$ can be approximated at a point $\vec{p} = \{a,b,c\}$ for unknown $\vec{q} = \{u,v,w\}$ by its Jacobian $\mathbf{J}(\vec{p})$ and a translation

$$G(\vec{q}) = G(\vec{p}) + \mathbf{J}(\vec{p})(\vec{q} - \vec{p})$$

Since the Jacobian represents the linear part of the transformation, its inverse transpose can be used to transform the normal:

$$\begin{aligned}\hat{N}(u,v,w)' &= \mathbf{J}(u,v,w)^{-1\top}\hat{N}(u,v,w) \\ &= -\mathbf{J}(u,v,w)^{-1\top}\frac{\nabla D(u,v,w)}{\|\nabla D(u,v,w)\|}\end{aligned}$$

Note that after the transformation, the magnitude of the normal might not be 1.

B. Distance Transforms and Deformations

Euclidean distance transforms can be applied to mesostructure shape data to produce a reprentation for fast ray traversal [Cohen and Sheffer 1994]. Given a mesostructure surface in a shell volume S_M, a texture-space unsigned distance field volume $\mathrm{DT}(u,v,w)$ can be defined as

$$\mathrm{DT}(u,v,w) = \min_{\vec{p}_M \in S_M} (\|(u,v,w) - \vec{p}_M\|). \tag{2}$$

A signed distance field can be obtained for orientable surfaces without boundaries by checking if the closest point is above or below the surface, which can be computed by checking the sign of the dot product of the surface normal and the distance vector.

Given a warp function $W(u,v,w)$ of the volume (e.g. the shell mapping $G(u,v,w)$), Equation (2) becomes:

$$\mathrm{DT_W}(u,v,w) = \min_{\vec{p}_M \in S_M} (\|W(u,v,w) - W(\vec{p}_M)\|).$$

In order for $\mathrm{DT_W}(u,v,w)$ to be equivalent to $\mathrm{DT}(u,v,w)$, W needs to be an isometric transformation. Given the generally non-affine nature of the deformation of shell space volumes, it is very unlikely that this requirement will be fulfilled in a per-face (prism or hexahedron) basis, so distance fields cannot be used for deformable surfaces without introducing distortions. This can be seen with a simple example in Figure 11.

The distance field of a grass texture, computed in texture space, is mapped on a quad. Sampling the top-right corner point would result in a safe distance to traverse. When the quad is undeformed, there are no errors. When the quad is deformed, the texture space deforms as well, since it is defined by coordinates at the quad's endpoints. In this deformed space, distances are not isotropic anymore, and, compared to the original space, they can be either too conservative, requiring more steps, or too large, skipping opaque regions in the distance field. Marching directly in texture space would eliminate this problem, but, in that space, rays would need to curve to avoid more artifacts, as seen in Figure 10.

C. Parametric Mapping

Following the format in [Heckbert 1989], trilinear mapping for a cuboid volume p_{xyz}, $\{x,y,z\} \in \{0,1\}$, can be expressed in a parametric form as follows:

$$\begin{pmatrix} P_x & P_y & P_z \end{pmatrix} = \begin{pmatrix} stq & sq & tq & st & s & q & t & 1 \end{pmatrix} \begin{pmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \\ D_x & D_y & D_z \\ E_x & E_y & E_z \\ F_x & F_y & F_z \\ G_x & G_y & G_z \\ H_x & H_y & H_z \end{pmatrix}.$$

where the values (s,t) represent the bilinear parametric coordinates of top and bottom quads, and q represents the intepolant value between these quads. The coefficients can be obtained by

$$\begin{pmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \\ D_x & D_y & D_z \\ E_x & E_y & E_z \\ F_x & F_y & F_z \\ G_x & G_y & G_z \\ H_x & H_y & H_z \end{pmatrix} = A \begin{pmatrix} p000_x & p000_y & p000_z \\ p100_x & p100_y & p100_z \\ p010_x & p010_y & p010_z \\ p110_x & p110_y & p110_z \\ p001_x & p001_y & p001_z \\ p101_x & p101_y & p101_z \\ p011_x & p011_y & p011_z \\ p111_x & p111_y & p111_z \end{pmatrix}$$

using the matrix

$$A = \begin{pmatrix} -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The above applies for extruded quadrilaterals. For prism volumes, defined by triangles $P_{ABC_{bot}}$ and $P_{ABC_{top}}$, the mapping is simpler:

$$\begin{pmatrix} P_x & P_y & P_z \end{pmatrix} = \begin{pmatrix} st & tq & s & t & q & 1 \end{pmatrix} \begin{pmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \\ D_x & D_y & D_z \\ E_x & E_y & E_z \\ F_x & F_y & F_z \end{pmatrix}$$

where the values (s,t) represent the barycentric coordinates of top and bottom triangles, and q represents the intepolant value between these triangles. The coefficients can be obtained by

$$\begin{pmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \\ D_x & D_y & D_z \\ E_x & E_y & E_z \\ F_x & F_y & F_z \end{pmatrix} = \mathbf{A} \begin{pmatrix} p_{A_{bot}x} & p_{A_{bot}y} & p_{A_{bot}z} \\ p_{B_{bot}x} & p_{B_{bot}y} & p_{B_{bot}z} \\ p_{C_{bot}x} & p_{C_{bot}y} & p_{C_{bot}z} \\ p_{A_{top}x} & p_{A_{top}y} & p_{A_{top}z} \\ p_{B_{top}x} & p_{B_{top}y} & p_{B_{top}z} \\ p_{C_{top}x} & p_{C_{top}y} & p_{C_{top}z} \end{pmatrix}$$

using the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & -1 & -1 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Both mappings exhibit C^0 continuity with adjacent volumes, since they do not take into account any neighboring information.

Acknowledgements

This work was funded by the EPSRC Centre for Digital Entertainment and Disney Research. Images in Figure 2 were obtained from Wikipedia. Thanks to deviantART artist Pablo An-

dreetta for providing the image in Figure 3. Thanks to Stefan Jeschke and Manuel Oliveira for providing source code for SSm/CSm and MLRm, respectively.

References

BECKER, B., AND MAX, N. 1993. Smooth transitions between bump rendering algorithms. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, SIGGRAPH '93, 183–190. http://doi.acm.org/10.1145/166117.166141. 29

BENSON, D., AND DAVIS, J. 2002. Octree textures. *ACM Trans. Graph. 21*, 3, 785–790. http://doi.acm.org/10.1145/566654.566652. 29

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, ACM, New York, NY, USA, SGP '04, 41–44. http://doi.acm.org/10.1145/1057432.1057437. 27

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1, 213–230. http://dx.doi.org/10.1109/TVCG.2007.1054. 45

BRODERSEN, A., MUSETH, K., PORUMBESCU, S., AND BUDGE, B. 2007. Geometric texturing using level sets. *IEEE Transactions on Visualization and Computer Graphics*, 277–288. http://dx.doi.org/10.1109/TVCG.2007.70408. 18, 21, 37, 38, 39

BRUNETON, E., AND NEYRET, F. 2012. A survey of nonlinear prefiltering methods for efficient and accurate surface shading. *IEEE Transactions on Visualization and Computer Graphics 18*, 2, 242–260. http://dx.doi.org/10.1109/TVCG.2011.81. 27, 29

BURLEY, B., AND LACEWELL, D. 2008. Ptex: Per-face texture mapping for production rendering. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, EGSR'08, 1155–1164. http://dx.doi.org/10.1111/j.1467-8659.2008.01253.x. 25

CATMULL, E. E. 1974. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis. The University of Utah, AAI7504786, www.pixartouchbook.com/storage/catmull_thesis.pdf. 19

CHEN, Y., AND CHANG, C. 2008. A prism-free method for silhouette rendering in inverse displacement mapping. *Computer Graphics Forum 27*, 7, 1929–1936. http://dx.doi.org/10.1111/j.1467-8659.2008.01341.x. 40

COHEN, D., AND SHEFFER, Z. 1994. Proximity clouds - an acceleration technique for 3d grid traversal. *The Visual Computer 11*, 1, 27–38. http://dx.doi.org/10.1007/BF01900697. 51

COOK, R., MAX, N., SILVA, C., AND WILLIAMS, P. 2004. Image-space visibility ordering for cell projection volume rendering of unstructured data. *IEEE Transactions on Visualization and Computer Graphics 10*, 6, 695–707. http://dx.doi.org/10.1109/TVCG.2004.45. 41

CRASSIN, C., NEYRET, F., LEFEBVRE, S., AND EISEMANN, E. 2009. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, I3D '09, 15–22. http://doi.acm.org/10.1145/1507149.1507152. 19, 30

DANA, K., VAN GINNEKEN, B., NAYAR, S., AND KOENDERINK, J. 1999. Reflectance and texture of real-world surfaces. *ACM Trans. Graph. 18*, 1, 1–34. http://doi.acm.org/10.1145/300776.300778. 30

DEBRY, D. G., GIBBS, J., PETTY, D. D., AND ROBINS, N. 2002. Painting and rendering textures on unparameterized models. ACM, New York, NY, USA, vol. 21, 763–768. http://doi.acm.org/10.1145/566654.566649. 29

DECAUDIN, P., AND NEYRET, F. 2009. Volumetric billboards. *Computer Graphics Forum 28*, 8, 2079–2089. http://dx.doi.org/10.1111/j.1467-8659.2009.01354.x. 18, 21, 28, 36, 39, 43, 44, 45

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 85–94. http://doi.acm.org/10.1145/280814.280826. 25

DONNELLY, W. 2005. Per-pixel displacement mapping with distance functions. *GPU Gems 2*, 2. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter08.html. 21, 28, 41, 45

DUFORT, J., LEBLANC, L., AND POULIN, P. 2005. Interactive rendering of meso-structure surface details using semi-transparent 3d textures. In *Proc. Vision, Modeling, and Visualization*, Citeseer, 399–406. http://www.iro.umontreal.ca/~dufortjf/vmv2005.pdf. 21, 28, 36, 39, 41, 45

EBERT, D., MUSGRAVE, F., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and modeling: a procedural approach*. Morgan Kaufmann. 26

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '95, 173–182. http://doi.acm.org/10.1145/218380.218440. 32

ELBER, G. 2005. Geometric texture modeling. *IEEE Comput. Graph. Appl. 25*, 4 (July), 66–76. http://dx.doi.org/10.1109/MCG.2005.79. 32

FATAHALIAN, K., LUONG, E., BOULOS, S., AKELEY, K., MARK, W. R., AND HANRAHAN, P. 2009. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proceedings of the Conference on High Performance Graphics 2009*, ACM, New York, NY, USA, HPG '09, 59–68. http://doi.acm.org/10.1145/1572769.1572780. 32

FATAHALIAN, K., BOULOS, S., HEGARTY, J., AKELEY, K., MARK, W. R., MORETON, H., AND HANRAHAN, P. 2010. Reducing shading on gpus using quad-fragment merging. *ACM Trans. Graph. 29*, 4 (July), 67:1–67:8. http://doi.acm.org/10.1145/1778765.1778804. 32

FILIP, J., AND HAINDL, M. 2009. Bidirectional texture function modeling: A state of the art survey. *IEEE Trans. Pattern Anal. Mach. Intell. 31*, 11 (Nov.), 1921–1940. http://dx.doi.org/10.1109/TPAMI.2008.246. 20, 27, 30

FOURNIER, A. 1992. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, 45–52. http://www.iro.umontreal.ca/~poulin/fournier/papers/Fournier-1992-NDF/Fournier-1992-NDFMS.pdf. 29

GILET, G., AND DISCHLER, J. 2009. A Framework for Interactive Hypertexture Modelling. In *Computer Graphics Forum*, vol. 28, Wiley Online Library, 2229–2243. http://dx.doi.org/10.1111/j.1467-8659.2009.01436.x. 21, 27

GLASSNER, A. S. 1994. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 22

GOBBETTI, E., MARTON, F., AND IGLESIAS GUITIÁN, J. 2008. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer 24*, 7, 797–806. http://vic.crs4.it/vic/cgi-bin/bib-page.cgi?id='Gobbetti:2008:SGR'. 30

HAN, C., SUN, B., RAMAMOORTHI, R., AND GRINSPUN, E. 2007. Frequency domain normal map filtering. ACM, New York, NY, USA, vol. 26. http://doi.acm.org/10.1145/1276377.1276412. 29

HART, J. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10, 527–545. http://www.eecs.wsu.edu/~hart/papers/zeno.ps.gz. 41

HECKBERT, P. S. 1989. Fundamentals of texture mapping and image warping. Tech. Rep. UCB/CSD-89-516, EECS Department, University of California, Berkeley, Jun. http://www.eecs.berkeley.edu/Pubs/TechRpts/1989/5504.html. 33, 51

HEITZ, E., NOWROUZEZAHRAI, D., POULIN, P., AND NEYRET, F. 2013. Filtering color mapped textures and surfaces. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '13, 129–136. http://doi.acm.org/10.1145/2448196.2448217. 28

HIRCHE, J., EHLERT, A., GUTHE, S., AND DOGGETT, M. 2004. Hardware accelerated per-pixel displacement mapping. In *Proceedings of the 2004 Graphics Interface Conference*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, GI '04, 153–158. http://dl.acm.org/citation.cfm?id=1006058.1006077. 21, 35, 36, 41

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: Theory and practice video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 Courses*, ACM, New York, NY, USA, SIGGRAPH '07. http://doi.acm.org/10.1145/1281500.1281510. 25

HU, L., SANDER, P. V., AND HOPPE, H. 2010. Parallel view-dependent level-of-detail control. *IEEE Transactions on Visualization and Computer Graphics 16*, 5 (Sept.), 718–728. http://dx.doi.org/10.1109/TVCG.2009.101. 32

IGLESIAS GUITIÁN, J., GOBBETTI, E., AND MARTON, F. 2010. View-dependent exploration of massive volumetric models on large-scale light field displays. *Vis. Comput. 26*, 6-8 (June), 1037–1047. http://dx.doi.org/10.1007/s00371-010-0453-y. 30

JESCHKE, S., MANTLER, S., AND WIMMER, M. 2007. Interactive smooth and curved shell mapping. 351–360. http://dx.doi.org/10.2312/EGWR/EGSR07/351-360. 21, 28, 36, 38, 39, 40, 41, 42, 45

KAJIYA, J. T., AND KAY, T. L. 1989. Rendering fur with three dimensional textures. ACM, New York, NY, USA, vol. 23, 271–280. http://doi.acm.org/10.1145/74334.74361. 34

KAJIYA, J. T. 1985. Anisotropic reflection models. ACM, New York, NY, USA, vol. 19, 15–21. http://doi.acm.org/10.1145/325165.325167. 29

KNISS, J., PREMOZE, S., HANSEN, C., AND EBERT, D. 2002. Interactive translucent volume rendering and procedural modeling. In *Proceedings of the Conference on Visualization '02*, IEEE Computer Society, Washington, DC, USA, VIS '02, 109–116. http://dl.acm.org/citation.cfm?id=602099.602114. 27

KOENDERINK, J. J., AND VAN DOORN, A. J. 1996. Illuminance texture due to surface mesostructure. *J. Opt. Soc. Am. A 13*, 3 (Mar), 452–463. http://josaa.osa.org/abstract.cfm?URI=josaa-13-3-452. 22

KONIARIS, C., COSKER, D., YANG, X., MITCHELL, K., AND MATTHEWS, I. 2013. Real-time content-aware texturing for deformable surfaces. 11:1–11:10. http://doi.acm.org/10.1145/2534008.2534016. 50

KOVACS, D., MITCHELL, J., DRONE, S., AND ZORIN, D. 2009. Real-time creased approximate subdivision surfaces. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '09, 155–160. http://doi.acm.org/10.1145/1507149.1507174. 25

KURZION, Y., AND YAGEL, R. 1995. Space deformation using ray deflectors. In *Rendering Techniques âĂŹ95*, Springer Vienna, P. Hanrahan and W. Purgathofer, Eds., Eurographics, 21–30. http://dx.doi.org/10.1007/978-3-7091-9430-0_3. 40

LAGAE, A., LEFEBVRE, S., COOK, R., DEROSE, T., DRETTAKIS, G., EBERT, D., LEWIS, J., PERLIN, K., AND ZWICKER, M. 2010. A survey of procedural noise functions. In *Computer Graphics Forum*, vol. 29, Wiley Online Library, 2579–2600. http://dx.doi.org/10.1111/j.1467-8659.2010.01827.x. 20, 27

LAI, Y.-K., HU, S.-M., GU, D. X., AND MARTIN, R. R. 2005. Geometric texture synthesis and transfer via geometry images. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, ACM, New York, NY, USA, SPM '05, 15–26. http://doi.acm.org/10.1145/1060244.1060248. 27

LAINE, S., AND KARRAS, T. 2010. Efficient sparse voxel octrees. 55–63. http://doi.acm.org/10.1145/1730804.1730814. 30

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 85–94. http://dx.doi.org/10.1145/344779.344829. 25

LEFEBVRE, S., HORNUS, S., NEYRET, F., ET AL. 2005. Octree textures on the gpu. *GPU gems 2*, 595–613. http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter37.html. 30

LEFOHN, A. E., SENGUPTA, S., KNISS, J., STRZODKA, R., AND OWENS, J. D. 2006. Glift: Generic, efficient, random-access gpu data structures. *ACM Trans. Graph. 25*, 1 (Jan.), 60–99. http://doi.acm.org/10.1145/1122501.1122505. 30

LOOP, C., SCHAEFER, S., NI, T., AND CASTAÑO, I. 2009. Approximating subdivision surfaces with gregory patches for hardware tessellation. 151:1–151:9. http://doi.acm.org/10.1145/1661412.1618497. 25

MCDONALD, JR, J., AND BURLEY, B. 2011. Per-face texture mapping for real-time rendering. In *ACM SIGGRAPH 2011 Studio Talks*, ACM, New York, NY, USA, SIGGRAPH '11, 3:1–3:1. http://doi.acm.org/10.1145/2037703.2037706. 25

MEYER, A., AND NEYRET, F. 1998. Interactive volumetric textures. In *Rendering techniques' 98: proceedings of the Eurographics Workshop in Vienna, Austria, June 29-July 1, 1998*, Springer Verlag Wien, 157. http://www-imagis.imag.fr/Publications/1998/MN98b/EGWR98.ps.gz. 34, 43

MILLER, C. M., AND JONES, M. W. 2005. Texturing and hypertexturing of volumetric objects. In *Proceedings of the Fourth Eurographics / IEEE VGTC Conference on Volume Graphics*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, VG'05, 117–125. http://dx.doi.org/10.2312/VG/VG05/117-125. 27

NA, K.-G., AND JUNG, M.-R. 2008. Curved ray-casting for displacement mapping in the gpu. 348–357. http://dl.acm.org/citation.cfm?id=1785794.1785832. 40

NEYRET, F. 1995. A general and multiscale model for volumetric textures. In *Graphics Interface*, 83–91. http://www.graphicsinterface.org/pre1996/95-Neyret.ps.gz. 34

NEYRET, F. 1996. Local illumination in deformed space. *Rapports de recherche- INRIA*. http://hal.inria.fr/inria-00073835. 40

NEYRET, F. 1998. Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan.), 55–70. http://dx.doi.org/10.1109/2945.675652. 29

NIESSNER, M., AND LOOP, C. 2013. Analytic displacement mapping using hardware tessellation. *ACM Trans. Graph. 32*, 3 (July), 26:1–26:9. http://doi.acm.org/10.1145/2487228.2487234. 25

NIESSNER, M., LOOP, C., MEYER, M., AND DEROSE, T. 2012. Feature-adaptive gpu rendering of catmull-clark subdivision surfaces. *ACM Trans. Graph. 31*, 1 (Feb.), 6:1–6:11. http://doi.acm.org/10.1145/2077341.2077347. 25

OLANO, M., AND BAKER, D. 2010. Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '10, 181–188. http://doi.acm.org/10.1145/1730804.1730834. 29

OLANO, M., AND NORTH, M. 1997. Normal distribution mapping. *Univ. of North Carolina Computer Science Technical Report*, 97–041. http://www.cs.unc.edu/~olano/papers/ndm/. 29

OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 359–368. http://dx.doi.org/10.1145/344779.344947. 28

PENG, J., KRISTJANSSON, D., AND ZORIN, D. 2004. Interactive modeling of topologically complex geometric detail. 635–643. http://doi.acm.org/10.1145/1186562.1015773. 18, 21, 28, 34, 37, 38, 39, 43, 45

PERLIN, K., AND HOFFERT, E. M. 1989. Hypertexture. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '89, 253–262. http://doi.acm.org/10.1145/74333.74359. 26

PERLIN, K. 1985. An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3 (July), 287–296. http://doi.acm.org/10.1145/325165.325247. 26

PIETRONI, N., CIGNONI, P., OTADUY, M., AND SCOPIGNO, R. 2010. Solid-texture synthesis: A survey. *IEEE Comput. Graph. Appl. 30*, 4 (July), 74–89. http://dx.doi.org/10.1109/MCG.2009.153. 27

POLICARPO, F., AND OLIVEIRA, M. M. 2006. Relief mapping of non-height-field surface details. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '06, 55–62. http://doi.acm.org/10.1145/1111411.1111422. 18, 21, 28, 42, 45

PORUMBESCU, S. D., BUDGE, B., FENG, L., AND JOY, K. I. 2005. Shell maps. 626–633. http://doi.acm.org/10.1145/1186822.1073239. 21, 24, 36, 37, 39, 40, 41, 44, 48

QUILEZ, I. 2008. Rendering worlds with two triangles with raytracing on the gpu in 4096 bytes. NVSCENE. 46

RITSCHE, N. 2006. Real-time shell space rendering of volumetric geometry. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, ACM, New York, NY, USA, GRAPHITE '06, 265–274. http://doi.acm.org/10.1145/1174429.1174477. 21, 28, 36, 39, 41, 45

SATHERLEY, R., AND JONES, M. W. 2002. Hypertexturing complex volume objects. *The Visual Computer 18*, 4, 226–235. http://dx.doi.org/10.1007/s003710100143. 27

SCHILLING, A. 1997. Towards real-time photorealistic rendering: Challenges and solutions. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, ACM, New York, NY, USA, HWWS '97, 7–15. http://doi.acm.org/10.1145/258694.258701. 29

SZIRMAY-KALOS, L., AND UMENHOFFER, T. 2008. Displacement mapping on the gpu - state of the art. *Computer Graphics Forum 27*, 6, 1567–1592. http://dx.doi.org/10.1111/j.1467-8659.2007.01108.x. 20, 23

TAN, P., LIN, S., QUAN, L., GUO, B., AND SHUM, H.-Y. 2005. Multiresolution reflectance filtering. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGSR'05, 111–116. http://dx.doi.org/10.2312/EGWR/EGSR05/111-116. 29

TOKSVIG, M. 2005. Mipmapping normal maps. *Journal of Graphics, GPU, & Game Tools 10*, 3, 65–71. http://dx.doi.org/10.1080/2151237X.2005.10129203. 29

WANG, L., WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2003. View-dependent displacement mapping. *ACM Trans. Graph. 22*, 3 (July), 334–339. http://doi.acm.org/10.1145/882262.882272. 21, 31, 43, 45

WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2004. Generalized displacement maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGSR'04, 227–233. http://dx.doi.org/10.2312/EGWR/EGSR04/227-233. 21, 31, 35, 39, 43, 44, 45

WANG, J., TONG, X., SNYDER, J., CHEN, Y., GUO, B., AND SHUM, H.-Y. 2005. Capturing and rendering geometry details for btf-mapped surfaces. *The Visual Computer 21*, 8-10, 559–568. http://dx.doi.org/10.1007/s00371-005-0318-y. 21, 28, 43, 45

WESTIN, S. H., ARVO, J. R., AND TORRANCE, K. E. 1992. Predicting reflectance functions from complex surfaces. *SIGGRAPH Comput. Graph. 26*, 2 (July), 255–264. http://doi.acm.org/10.1145/142920.134075. 22

WILLIAMS, L. 1983. Pyramidal parametrics. ACM, New York, NY, USA, vol. 17, 1–11. http://doi.acm.org/10.1145/964967.801126. 29

ZHOU, K., HUANG, X., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H.-Y. 2006. Mesh quilting for geometric texture synthesis. *ACM Trans. Graph. 25*, 3 (July), 690–697. http://doi.acm.org/10.1145/1141911.1141942. 21, 27, 36, 38, 39

Charalampos Koniaris, Darren Cosker, Xiaosong Yang, and Kenny Mitchell, Texture Mapping Techniques for Volumetric Mesostructure, *Journal of Computer Graphics Techniques (JCGT)*, vol. 3, no. 2, 18–59, 2014
http://jcgt.org/published/0001/03/02/

Received:	2013-09-25		
Recommended:	2013-11-22	Corresponding Editor:	Reynold Bailey
Published:	2014-02-27	Editor-in-Chief:	Morgan McGuire